


```
000000  TTTTTTTTTT  SSSSSSSS  CCCCCCCC  CCCCCCCC  BBBB88888
000000  TTTTTTTTTT  SSSSSSSS  CCCCCCCC  CCCCCCCC  BBBB88888
00      TT      SS      CC      CC      BB      BB
00      TT      SS      CC      CC      BB      BB
00      TT      SS      CC      CC      BB      BB
00      TT      SS      CC      CC      BB      BB
00      TT      SSSSSS  CC      CC      BBBB88888
00      TT      SSSSSS  CC      CC      BBBB88888
00      TT      SS      CC      CC      BB      BB
00      TT      SS      CC      CC      BB      BB
00      TT      SS      CC      CC      BB      BB
000000  TT      SSSSSSSS  CCCCCCCC  CCCCCCCC  BBBB88888
000000  TT      SSSSSSSS  CCCCCCCC  CCCCCCCC  BBBB88888
                                ....
                                ....
                                ....
                                ....
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLL  IIIIII  SSSSSSSS
```



```
0001 0 MODULE OTSS$CCB (
0002 0 IDENT = '1-057'
0003 0 ) =
0004 1 BEGIN
0005 1
0006 1 *****
0007 1 *
0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0010 1 * ALL RIGHTS RESERVED.
0011 1 *
0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0017 1 * TRANSFERRED.
0018 1 *
0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0021 1 * CORPORATION.
0022 1 *
0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0025 1 *
0026 1 *
0027 1 *****
0028 1
0029 1 ++
0030 1 FACILITY: language support library
0031 1
0032 1 ABSTRACT:
0033 1
0034 1 This module supports pushing and popping of the CCB, the
0035 1 common control block for the I/O part of the RTL. Currently,
0036 1 only BASIC uses this module, since FORTRAN does its own
0037 1 manipulations.
0038 1
0039 1 ENVIRONMENT: User mode, AST level or not or mixed
0040 1
0041 1 AUTHOR: Thomas N. Hastings, CREATION DATE: 01-June-77
0042 1
0043 1 MODIFIED BY:
0044 1
0045 1 Thomas N. Hastings, 01-June-77: VERSION 01
0046 1 01 - original
0047 1 0-26 - Set RMS RAB$V_UIF bit TNH 19-SEP-77
0048 1 0-27 - Set RMS RAB$V_TPT bit (truncate on sequential $PUT not at EOF TNH 24-SEP-77
0049 1 0-28 - Use FOR$$SIG_NO_LUB since no LUB. TNH 24-SEP-77
0050 1 0-30 - Set RAB bits for read-ahead, write-behind, locate mode JMT 21-OCT-77
0051 1 0-31 - Use FOR$K_abcnoxyz as EXTERNAL LITERALS. TNH 27-Oct-77
0052 1 0-32 - Made second arg optional. TNH 9-Nov-77
0053 1 0-33 - Use OTSS$FATINTERR. TNH 01-Dec-77
0054 1 0-34 - Clear FAB after call to LIB$GET_VM. TNH 9-Dec-77
0055 1 0-35 - Call FOR$$SIG_FATINT. TNH 30-Dec-77
0056 1 0-36 - Have CB_POP signal FATINT if LUB not active;
0057 1 Add routine CB_CND_POP to conditionally pop if LUB active.
```

```
! Push, Pop, Allocate, and deallocate LUB/ISB/RAB
! File: OTSCCB.B32 Edit: FM1057
```



```
58 0058 1 otherwise NO-OP (OTS exit handler calls this). JMT 10-Jan-78
59 0059 1 0-37 - Remove CB_CND_POP; I didn't really want it, anyway... JMT 11-Jan-78
60 0060 1 0-37 - Global register CCB. JMT 8-Apr-78
61 0061 1 0-39 - Change to STARLET library. DGP 20-Apr-78
62 0062 1 0-40 - Change REQUIRE files for VAX system build. DGP 28-Apr-78
63 0063 1 0-41 - Change STARLET to RTLSTARLE to avoid conflicts. DGP 1-May-78
64 0064 1 0-42 - Make JSB linkage. TNH 19-May-78
65 0065 1 0-46 - Use FOR$$GET_VM with new optional 2nd arg. TNH 21-May-78
66 0066 1 0-47 - Remove setting ISB to -1. TNH 30-May-78.
67 0067 1 0-48 - Add sanity check of data base. TNH 10-June-78
68 0068 1 0-49 - Add call to FOR$$SIG_DATCOR. TNH 10-June-78
69 0069 1 0-50 - Add FOR$$CB_GET entry for non-shared access to OTS$$A_CUR_LUB. TNH 2-Aug-78
70 0070 1 0-52 - Fix AST re-entrant timing hole. TNH 9-Aug-78
71 0071 1 0-53 - Change file name to FORCB.B32, and change the names of the
72 0072 1 REQUIRE files similarly. JBS 14-NOV-78
73 0073 1 1-001 - Update version number and copyright notice. JBS 16-NOV-78
74 0074 1 1-002 - Change LUB$B_LUN to LUB$W_LUN. JBS 05-DEC-78
75 0075 1 1-003 - Change REQUIRE file names from FOR... to OTS... JBS 07-DEC-78
76 0076 1 1-004 - Include TNH's version, which uses a bit table to provide
77 0077 1 AST re-entrancy. JBS 11-DEC-78
78 0078 1 1-005 - Remove REQUIRE of OTSMAC; not needed. JBS 11-DEC-78
79 0079 1 1-006 - Add FOR$$CB_NEXT, which gets the next LUN for the CLOSE loop
80 0080 1 in FOROPEN.B32. JBS 11-DEC-78
81 0081 1 1-007 - Fix coding errors in FOR$$CB_NEXT and make OTS$$AA_LUB_TAB
82 0082 1 OWN. JBS 18-DEC-78
83 0083 1 1-008 - Change file and module name to OTSCB and add specialized
84 0084 1 BASIC entry points. This is in preparation for recursive
85 0085 1 I/O. JBS 29-DEC-78
86 0086 1 1-009 - Add BAS$$CB_CLEANUP. JBS 29-DEC-78
87 0087 1 1-010 - Add recursive I/O for BASIC. JBS 08-JAN-1979
88 0088 1 1-011 - Divide into three modules: OTSCCB, FORCB and BASCB. This
89 0089 1 module, OTSCCB, contains the language-independent code.
90 0090 1 JBS 09-JAN-1979
91 0091 1 1-012 - Restore OTS$$A_CUR_LUB and set I/O Active when popping.
92 0092 1 JBS 15-JAN-1979
93 0093 1 1-013 - Fix up some complex cases of popping recursive I/O.
94 0094 1 JBS 15-JAN-1979
95 0095 1 1-014 - Fix an error in calling LIB$STOP. JBS 16-JAN-1979
96 0096 1 1-015 - Push and Pop the RMS timeout field in the RAB. JBS 16-JAN-1979
97 0097 1 1-016 - Use the DEALLOC bit in the LUB to interlock deallocation of
98 0098 1 the LUB/ISB/RAB rather than disabling interrupts, and be
99 0099 1 cleverer in other places so that interrupts need never be
100 0100 1 disabled. JBS 23-JAN-1979
101 0101 1 1-017 - Don't clear OTS$$V_IOINPROG if the LUN we just popped is the same
102 0102 1 unit we just finished using. (This is the most common case
103 0103 1 of recursive I/O.) JBS 24-JAN-1979
104 0104 1 1-018 - But if there is no popped unit, do clear OTS$$V_IOINPROG. (This is
105 0105 1 the most common case of non-recursive I/O!). JBS 24-JAN-1979
106 0106 1 1-019 - Divide into more internal subroutines in an attempt to speed
107 0107 1 up the common pushing and popping cases by avoiding the
108 0108 1 saving of unnecessary registers. JBS 25-JAN-1979
109 0109 1 1-020 - Change linkage for OTS$PUSH_CCB to JSB CB PUSH and for
110 0110 1 OTS$POP_CCB to JSB CB_POP. JBS 25-JAN-1979
111 0111 1 1-021 - Clear length of prompt buffer when pushing. JBS 26-JAN-1979
112 0112 1 1-022 - Remove OTS$CLEANUP_IO, we will clean I/O using a stack
113 0113 1 frame instead. JBS 26-JAN-1979
114 0114 1 1-023 - Change to double dollar signs since these entry points are
```


115 0115 1 not for use by users. JBS 26-JAN-1979
116 0116 1 1-024 - Deallocate the LUN after the LUB/ISB/RAB has been deallocated.
117 0117 1 Note that OPEN allocates it. JBS 26-JAN-1979
118 0118 1 1-025 - Make the table storage PIC, even though it is used by INSQUE
119 0119 1 and REMQUE instructions, by initializing it at run time.
120 0120 1 This requires disabling ASTs during initialization, but it is
121 0121 1 done only once per image activation. JBS 28-JAN-1979
122 0122 1 1-026 - Rearrange the order of some of the manipulations to make
123 0123 1 PUSH and POP really AST re-entrant. JBS 29-JAN-1979
124 0124 1 1-027 - Make these routines AST reentrant
125 0125 1 in the face of deallocation at AST level. JBS 31-JAN-1979
126 0126 1 1-028 - If LUB\$V_USER_RBUF is set, don't deallocate the record
127 0127 1 buffer, it belongs to the user! JBS 16-FEB-1979
128 0128 1 1-029 - Clear the buddy's buddy pointer, which points to us, when
129 0129 1 deallocating. JBS 16-FEB-1979
130 0130 1 1-030 - Print an error message if the ISB overlaps the LUB. This can
131 0131 1 happen if the LUB is extended but the ISB is not edited to
132 0132 1 reflect it. JBS 21-MAR-1979
133 0133 1 1-031 - Initialize LUB\$Q_BFA_QUEUE. JBS 05-APR-1979
134 0134 1 1-032 - Don't free the file name string unless it has been allocated
135 0135 1 in virtual memory. JBS 10-APR-1979
136 0136 1 1-033 - Don't free the record buffer unless it has been allocated.
137 0137 1 JBS 10-APR-1979
138 0138 1 1-034 - Free the compiled format, if allocated. SBL 27-Apr-1979
139 0139 1 1-035 - Set ISB\$W_FMT_LEN to zero on allocation. SBL 4-May-79
140 0140 1 1-036 - Change CASE off result of REMQUE to match what is
141 0141 1 actually given by that function. SBL 9-May-1979
142 0142 1 1-037 - Change required file name to OTSCCBREQ so as not to conflict
143 0143 1 with this module at system build time. SBL 10-May-1979
144 0144 1 1-038 - Move clearing of ISB\$W_FMT_LEN to allocation stage. SBL 14-May-1979
145 0145 1 1-039 - Fix bug in compiled format deallocation. SBL 15-May-1979
146 0146 1 1-040 - Fix another one. Length must be passed as address of a word!
147 0147 1 1-041 - We overlooked the REMQUE in DEALLOCATE. SBL 17-May-1979
148 0148 1 So, we have to construct a temp. SBL 17-May-1979
149 0149 1 1-042 - Clear ISB\$W_FMT_LEN during PUSH, so that POP won't try
150 0150 1 to deallocate the format prematurely. JBS 29-MAY-1979
151 0151 1 1-043 - Set up LUB\$A_BUDDY_PTR during allocate. JBS 30-MAY-1979
152 0152 1 1-044 - Make much of the data structure global so it can be
153 0153 1 referenced directly by FOR\$CCB. JBS 28-JUN-1979
154 0154 1 1-045 - Do an RMS \$WAIT if there is I/O active on the unit we
155 0155 1 are starting. JBS 25-JUL-1979
156 0156 1 1-046 - Don't make OTSS\$Q IO ACTIVE global. JBS 26-JUL-1979
157 0157 1 1-047 - Save the prompt buffer only if it really is a prompt buffer.
158 0158 1 If it is a key buffer, the key may be in read-only storage.
159 0159 1 JBS 09-AUG-1979
160 0160 1 1-048 - Move the global parts of the data base to OTSS\$CCB_DATA
161 0161 1 so this module need not be loaded if only FORTRAN programs
162 0162 1 are in the image. JBS 16-AUG-1979
163 0163 1 1-049 - Return CCB as 0 from POP to indicate deallocation. JBS 17-AUG-1979
164 0164 1 1-050 - Correct an error in a comment. JBS 10-SEP-1979
165 0165 1 1-051 - When deallocating, LUB\$A_BUF_BEG points to the buffer; in locate
166 0166 1 mode, LUB\$A_RBUF_ADR may point to RMS space. JBS 13-SEP-1979
167 0167 1 1-052 - Remove the references to ISB\$W_FMT_LEN; now done in FORCB.
168 0168 1 JBS 18-SEP-1979
169 0169 1 1-053 - Remove references to LUB\$Q_BFA_QUEUE; no longer used.
170 0170 1 JBS 18-SEP-1979
171 0171 1 1-054 - Correct a minor typo. JBS 24-OCT-1979

OTSS\$CCB
1-057

F 6
16-Sep-1984 01:22:30
14-Sep-1984 12:39:38

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]OTSCCB.B32;1

Page 4
(1)

```
: 172      0172 1 : 1-055 - Use the new UBF cell in the LUB, JBS 13-NOV-1979
: 173      0173 1 : 1-056 - Don't initialize LUB table entries in use by FORTRAN. JBS 14-JAN-1980
: 174      0174 1 : 1-057 - Take out clearing of RAB$B PSZ (put it in BAS$$IO_BEG)
: 175      0175 1 : to make locality consistent. FM 4-SEP-1980
: 176      0176 1 : --
: 177      0177 1 :
: 178      0178 1 : <BLF/PAGE>
```



```
180 0179 1 |
181 0180 1 | SWITCHES:
182 0181 1 |
183 0182 1 |
184 0183 1 | SWITCHES ADDRESSING_MODE (EXTERNAL = GENERAL, NONEXTERNAL = WORD_RELATIVE);
185 0184 1 |
186 0185 1 |
187 0186 1 | LINKAGES:
188 0187 1 |
189 0188 1 |
190 0189 1 | REQUIRE 'RTLIN:OTSLNK'; | Define LINKAGES
191 0618 1 |
192 0619 1 |
193 0620 1 | TABLE OF CONTENTS:
194 0621 1 |
195 0622 1 |
196 0623 1 | FORWARD ROUTINE
197 0624 1 | INITIALIZE : NOVALUE, | Set up the LUB table and the active queue
198 0625 1 | PUSH_FAKE : CALL CCB, | Push fake record
199 0626 1 | PUSH_ACTIVE : CALL CCB, | Push active LUB
200 0627 1 | ALLOCATE : CALL CCB, | Allocate LUB/ISB/RAB
201 0628 1 | OTSS$PUSH_CCB : JSB CB PUSH, | Get the CCB, push old use of it
202 0629 1 | DEALLOCATE : CALL CCB NOVALUE, | Deallocate LUB/ISB/RAB
203 0630 1 | POP_ACTIVE : CALL CCB NOVALUE, | Pop active LUN
204 0631 1 | OTSS$POP_CCB : JSB CB_POP NOVALUE; | Restore old use of CCB
205 0632 1 |
206 0633 1 |
207 0634 1 | INCLUDE FILES:
208 0635 1 |
209 0636 1 |
210 0637 1 | REQUIRE 'RTLML:OTSISB'; | get length of ISB
211 0805 1 |
212 0806 1 | REQUIRE 'RTLML:OTSLUB'; | get length of LUB
213 0946 1 |
214 0947 1 | REQUIRE 'RTLIN:RTLPSECT'; | Define DECLARE_PSECTs macro
215 1042 1 |
216 1043 1 | REQUIRE 'RTLIN:OTSCCBREQ'; | Define interface to OTSS$PUSH_CCB
217 1141 1 |
218 1142 1 | LIBRARY 'RTLSTARLE'; | STARLET library for macros and symbols
219 1143 1 |
220 1144 1 |
221 1145 1 | MACROS:
222 1146 1 |
223 1147 1 |
224 1148 1 | MACRO
225 M 1149 1 | TEST_LUB_ISB =
226 M 1150 1 | +
227 M 1151 1 | Give an error message if the ISB and the LUB overlap. Try to make the
228 M 1152 1 | message explicit enough to tell the maintainer exactly what to do, since
229 M 1153 1 | it will print only when the RTL is being modified by someone who does not
230 M 1154 1 | know about the LUB-ISB dependency, and therefore may need a lot of hand-
231 M 1155 1 | holding.
232 M 1156 1 | -
233 M 1157 1 |
234 M 1158 1 | %IF (LUB$K_NEG_BLN NEQ ISB$K_NEG_LUB)
235 M 1159 1 | %THEN
236 M 1160 1 |
```



```
237 M 1161 1 COMPILETIME
238 M 1162 1 VAL1 = -ISB$K_NEG_LUB,
239 M 1163 1 VAL2 = -LUB$K_NEG_BLN;
240 M 1164 1
241 M 1165 1 %ERROR (' LUB$K_NEG_BLN is not equal to ISB$K_NEG_LUB.',
242 M 1166 1 ' This probably means that the LUB has been extended',
243 M 1167 1 ' without editing the ISB to allow for it. Please edit file OTSISB.MDL, making the -F,B,',
244 M 1168 1 %NUMBER (VAL1), ' be -F,B,', %NUMBER (VAL2))
245 M 1169 1 %FI
246 M 1170 1
247 M 1171 1 %;
248 M 1172 1
249 M 1173 1
250 M 1174 1 EQUATED SYMBOLS:
251 M 1175 1
252 M 1176 1
253 M 1177 1 LITERAL
254 M 1178 1 K_TOTAL_CCB_LEN = LUB$K_LUB_LEN + ISB$K_ISB_LEN + RAB$C_BLN; ! length of LUB+ISB+RAB
255 M 1179 1
256 M 1180 1
257 M 1181 1 PSECT DECLARATIONS:
258 M 1182 1
259 M 1183 1 DECLARE_PSECTS (OTS); ! declare PSECTs for OTSS$ facility
260 M 1184 1
261 M 1185 1 OWN STORAGE:
262 M 1186 1
263 M 1187 1 +
264 M 1188 1 The following quadword is the header of the I/O active queue. Items
265 M 1189 1 are manipulated on this queue using the INSQUE and REMQUE instructions.
266 M 1190 1 -
267 M 1191 1
268 M 1192 1 OWN
269 M 1193 1 OTSSQ_IO_ACTIVE : VECTOR [2];
270 M 1194 1
271 M 1195 1
272 M 1196 1 EXTERNAL REFERENCES:
273 M 1197 1
274 M 1198 1
275 M 1199 1 EXTERNAL ROUTINE
276 M 1200 1 LIB$GET_VM, ! Allocate virtual memory
277 M 1201 1 LIB$FREE_VM, ! Deallocate virtual memory
278 M 1202 1 LIB$STOP : NOVALUE, ! Signal a fatal error
279 M 1203 1 OTSS$FREE_LUN; ! Deallocate a LUN
280 M 1204 1
281 M 1205 1 EXTERNAL LITERAL
282 M 1206 1 OTSS_FATINTERR : UNSIGNED (%BPVAL); ! condition value for FATAL INTERNAL ERROR
283 M 1207 1
284 M 1208 1 ! IN RUN-TIME LIBRARY error.
285 M 1209 1 +
286 M 1210 1 The following externals represent the global part of the CCB
287 M 1211 1 data base.
288 M 1212 1 -
289 M 1213 1
290 M 1214 1 EXTERNAL
291 M 1215 1 OTSS$V_CCB_INIT : VOLATILE, ! True if INIT done
292 M 1216 1 OTSS$A_LUB_TAB : VOLATILE OTSS$LUB_TAB ST !
293 M 1217 1 [-LUB$K_ILUN_MIN + LUB$K_LUN_MAX + 1, LUB$K_ILUN_MIN], ! Pointers to CCBs
```

:	294	1218	1	OTSS\$V_IOINPROG : VOLATILE BITVECTOR,	! True if LUN has I/O active
:	295	1219	1	OTSS\$A_CUR_LUB,	! The current LUB
:	296	1220	1	OTSS\$L_CUR_LUN,	! The current logical unit
:	297	1221	1	OTSS\$L_LVL_CTR;	! -1 = ILDE, 0 = 1 I/O in progress.
:	298	1222	1		
:	299	1223	1	BUILTIN	
:	300	1224	1	INSQUE,	! Insert an item in a queue
:	301	1225	1	REMQUE,	! Remove an item from a queue
:	302	1226	1	TESTBITSS,	! Test bit, set it, return true if it was set.
:	303	1227	1	TESTBITCC;	! Test bit, clear it, return true if it was clear.
:	304	1228	1		
:	305	1229	1	!<BLF/PAGE>	


```

: 307      1230 1  !+
: 308      1231 1  ! The following field set represents an item pushed onto the
: 309      1232 1  ! I/O Active list. It contains the ISB, the prompt buffer, the
: 310      1233 1  ! current size of the prompt buffer, and the timeout value from
: 311      1234 1  ! the RAB.
: 312      1235 1  !-
: 313      1236 1  !
: 314      1237 1  FIELD
: 315      1238 1  PUSH_ITEM =
: 316      1239 1  SET
: 317      1240 1  PUSH$A_NEXT = [0, 0, %BPVAL, 0],      ! Next item
: 318      1241 1  PUSH$A_PREV = [4, 0, %BPVAL, 0],      ! Previous item
: 319      1242 1  PUSH$L_STS = [8, 0, %BPVAL, 0],      ! RMS status
: 320      1243 1  PUSH$L_STV = [12, 0, %BPVAL, 0],      ! RMS extra status
: 321      1244 1  PUSH$W_LUN = [16, 0, 16, 1],      ! Logical unit number
: 322      1245 1  PUSH$B_PSZ = [18, 0, 8, 0],      ! Prompt buffer size
: 323      1246 1  PUSH$B_TMO = [19, 0, 8, 0],      ! The RMS timeout value
: 324      1247 1  PUSH$V_IO_ACT = [20, 0, 1, 0],      ! The I/O Active flag
: 325      1248 1  PUSH$V_FARE = [20, 1, 1, 0],      ! The "fake" flag
: 326      1249 1  PUSH$V_PMT = [20, 2, 1, 0],      ! Set if there is a prompt buffer.
: 327      1250 1  PUSH$T_PROMPT = [21, 0, 0, 0],      ! The prompt buffer
: 328      1251 1  PUSH$X_ISB = [LUB$K_PBUF_SIZ + 21, 0, 0, 0] ! The ISB
: 329      1252 1  TES;
: 330      1253 1
: 331      1254 1 LITERAL
: 332      1255 1  PUSH$K_LENGTH = 21 + LUB$K_PBUF_SIZ + ISB$K_ISB_LEN;      ! Number of bytes to allocate
: 333      1256 1
```

```

335 1257 1 ROUTINE INITIALIZE : NOVALUE =                ! Set up OWN storage
336 1258 1
337 1259 1 ++
338 1260 1 FUNCTIONAL DESCRIPTION:
339 1261 1
340 1262 1     Set up the LUB table, I/O Active queue and OTSS$L_CUR_LUN.
341 1263 1     The LUB table and I/O active queue must be set up at run time
342 1264 1     because they must be initialized with addresses, and this
343 1265 1     cannot be done at link time or they will cease to be position
344 1266 1     independent. They must be initialized with addresses because
345 1267 1     they are used by INSQUE and REMQUE to avoid disabling ASTs.
346 1268 1
347 1269 1 CALLING SEQUENCE:
348 1270 1
349 1271 1     IF (NOT .OTSS$V_CCB_INIT) THEN INITIALIZE ();
350 1272 1
351 1273 1 FORMAL PARAMETERS:
352 1274 1
353 1275 1     NONE
354 1276 1
355 1277 1 IMPLICIT INPUTS:
356 1278 1
357 1279 1     OTSS$AA_LUB_TAB
358 1280 1     OTSS$Q_TO_ACTIVE
359 1281 1     OTSS$V_IOINPROG
360 1282 1     OTSS$L_CUR_LUN
361 1283 1
362 1284 1 IMPLICIT OUTPUTS:
363 1285 1
364 1286 1     OTSS$AA_LUB_TAB
365 1287 1     OTSS$Q_TO_ACTIVE
366 1288 1     OTSS$V_IOINPROG
367 1289 1     OTSS$L_CUR_LUN
368 1290 1     OTSS$V_CCB_INIT
369 1291 1
370 1292 1 SIDE EFFECTS:
371 1293 1
372 1294 1     NONE
373 1295 1 --
374 1296 1
375 1297 2 BEGIN
376 1298 2
377 1299 2 LOCAL
378 1300 2 ++
379 1301 2 The following cell keeps track of whether or not ASTs were disabled
380 1302 2 when we were called.
381 1303 2 --
382 1304 2 AST_STATUS;
383 1305 2
384 1306 2 ++
385 1307 2 First disable ASTs. Then, if the initialization has not yet been
386 1308 2 done, do it. The initialization will have been done if an AST went
387 1309 2 off between the test of OTSS$V_CCB_INIT and this point.
388 1310 2 --
389 1311 2 AST_STATUS = $SETAST (ENBFLG = 0);
390 1312 2
391 1313 3 IF ( NOT .OTSS$V_CCB_INIT)
```



```

392      1314 2 THEN
393      1315 BEGIN
394      1316
395      1317 + We must do the initialization. First set the LUB table to be empty.
396      1318 Note that LUBs in use by FORTRAN are not touched. FORTRAN leaves the
397      1319 first longword non-zero for entries it is using.
398      1320 -
399      1321
400      1322 INCR LUN FROM LUB$K ILUN_MIN TO LUB$K LUN_MAX DO
401      1323 IF (.OTSS$AA_LUB_TAB [.LUN, 0] EQL 0) THEN
402      1324 OTSS$AA_LUB_TAB [.LUN, 0] = OTSS$AA_LUB_TAB [.LUN, 1] = OTSS$AA_LUB_TAB [.LUN, 0];
403      1325
404      1326 +
405      1327 Now make the I/O active queue empty.
406      1328 -
407      1329 OTSS$Q_IO_ACTIVE [0] = OTSS$Q_IO_ACTIVE [1] = OTSS$Q_IO_ACTIVE [0];
408      1330 +
409      1331 Mark that the initialization has been done, so it won't be done again.
410      1332 -
411      1333 OTSS$V_CCB_INIT = 1;
412      1334 END;
413      1335
414      1336 +
415      1337 If ASTs were enabled at entry, re-enable them.
416      1338 -
417      1339
418      1340 IF (.AST_STATUS EQL SS$_WASSET) THEN $SETAST (ENBFLG = 1);
419      1341
420      1342 RETURN;
421      1343 END;

```

! of routine INITIALIZE

.TITLE OTSS\$CCB
.IDENT \1-057\

.PSECT _OTSS\$DATA,NOEXE, PIC,2

00000 OTSS\$Q_IO_ACTIVE:

.BLKB 8

.EXTRN LIB\$GET_VM, LIB\$FREE_VM
.EXTRN LIB\$STOP, OTSS\$FREE_LUN
.EXTRN OTSS\$FATINTERR, OTSS\$V_CCB_INIT
.EXTRN OTSS\$AA_LUB_TAB
.EXTRN OTSS\$V_JOINPROG
.EXTRN OTSS\$A_CUR_LUB, OTSS\$L_CUR_LUN
.EXTRN OTSS\$L_LVL_CTR, SYS\$SETAST

.PSECT _OTSS\$CODE,NOWRT, SHR, PIC,2

003C 00000 INITIALIZE:

```

55 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5
54 00000000G 00 9E 00009 MOVAB SYS$SETAST, R5
53 00000000' EF 9E 00010 MOVAB OTSS$V_CCB_INIT, R4
7E D4 00017 CLRL -(SP)
65 01 FB 00019 CALLS #1, SYS$SETAST

```

: 1257

: 1311

OTSS\$CCB
1-057

M 6
16-Sep-1984 01:22:30 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:39:38 [LIBRTL.SRC]OTSCCB.B32;1

Page 11
(4)

	31	64	E8	0001C	BLBS	OTSS\$V_CCB_INIT, 3\$:	1313
	51	08	CE	0001F	MNEGL	#8, LUN	:	1322
	52	00000000G0041	7E	00022	1\$: MOVAQ	OTSS\$AA_LUB_TAB+64[LUN], R2	:	1323
		62	D5	0002A	TSTL	(R2)	:	
		0D	12	0002C	BNEQ	2\$:	
		00000000G0041	7F	0002E	PUSHAQ	OTSS\$AA_LUB_TAB+68[LUN]	:	1324
	9E	52	D0	00035	MOVL	R2, @ (SP)+	:	
	62	52	D0	00038	MOVL	R2, (R2)	:	
DF	51	00000077	8F	F3	2\$: AOBLEQ	#119, LUN, 1\$:	1323
	51	63	9E	00043	MOVAB	OTSS\$IO_ACTIVE, R1	:	1329
04	A3	51	D0	00046	MOVL	R1, OTSS\$IO_ACTIVE+4	:	
	63	51	D0	0004A	MOVL	R1, OTSS\$IO_ACTIVE	:	
	64	01	D0	0004D	MOVL	#1, OTSS\$V_CCB_INIT	:	1333
	09	50	D1	00050	3\$: CMPL	AST_STATUS, #9	:	1340
		05	12	00053	BNEQ	4\$:	
		01	DD	00055	PUSHL	#1	:	
	65	01	FB	00057	CALLS	#1, SYS\$SETAST	:	
		04	0005A	4\$: RET			:	1343

; Routine Size: 91 bytes, Routine Base: _OTSS\$CODE + 0000


```

423 1344 1 ROUTINE PUSH_FAKE : CALL_CCB =                ! Push a "fake" active record
424 1345 1
425 1346 1 !++
426 1347 1 FUNCTIONAL DESCRIPTION:
427 1348 1
428 1349 1     Push onto the I/O Active queue a place holder. This is to
429 1350 1     satisfy POP_ACTIVE when we can't actually push the CCB.
430 1351 1
431 1352 1 CALLING SEQUENCE:
432 1353 1
433 1354 1     CALL PUSH_FAKE ();
434 1355 1
435 1356 1 FORMAL PARAMETERS:
436 1357 1
437 1358 1     NONE
438 1359 1
439 1360 1 IMPLICIT INPUTS:
440 1361 1
441 1362 1     OTSS$Q_IO_ACTIVE
442 1363 1     OTSS$L_CUR_LUN
443 1364 1
444 1365 1 IMPLICIT OUTPUTS:
445 1366 1
446 1367 1     OTSS$Q_IO_ACTIVE           Holds previous I/O on this LUN
447 1368 1
448 1369 1 SIDE EFFECTS:
449 1370 1
450 1371 1     Calls LIB$GET_VM to get virtual memory.
451 1372 1 !--
452 1373 1
453 1374 2 BEGIN
454 1375 2
455 1376 2 EXTERNAL REGISTER
456 1377 2     CCB : REF BLOCK [, BYTE];
457 1378 2
458 1379 2 LOCAL
459 1380 2 !+
460 1381 2 Declare the pointer to the block to push.
461 1382 2 !-
462 1383 2     PUSH : REF BLOCK [PUSH$K_LENGTH, BYTE] FIELD (PUSH_ITEM),
463 1384 2     LUN;
464 1385 2
465 1386 2     LUN = .OTSS$L_CUR_LUN;
466 1387 2 !+
467 1388 2 Get virtual memory to hold the fake activation record.
468 1389 2 !-
469 1390 2 BEGIN
470 1391 2
471 1392 2 LOCAL
472 1393 2     GET_VM_RESULT;
473 1394 2
474 1395 2     GET_VM_RESULT = LIB$GET_VM (%REF (PUSH$K_LENGTH), PUSH);
475 1396 2
476 1397 2     IF ( NOT .GET_VM_RESULT ) THEN RETURN (OTSS$K_PUSH_FAIL);
477 1398 2
478 1399 2 END;
479 1400 2 !+
```

```

: 480      1401 2 | Copy the old LUN into the fake record, and mark it as fake.
: 481      1402 2 |
: 482      1403 2 |   PUSH [PUSH$W_LUN] = .LUN;
: 483      1404 2 |   PUSH [PUSH$V_FAKE] = 1;
: 484      1405 2 |
: 485      1406 2 | + Put this item on the I/O Active list.
: 486      1407 2 |
: 487      1408 2 |   INSQUE (.PUSH, OTSSQ_IO_ACTIVE);
: 488      1409 2 |
: 489      1410 2 | + We also set OTSS$L_CUR_LUN to LUB$K_LUN_MAX+1 to prevent an
: 490      1411 2 |   AST from pushing that LUB again. An extra push before this point
: 491      1412 2 |   does not cause any harm (only wastes a little time).
: 492      1413 2 |
: 493      1414 2 |   OTSS$L_CUR_LUN = LUB$K_LUN_MAX + 1;
: 494      1415 2 |   RETURN (OTSS$K_PUSH_OK);
: 495      1416 1 | END;

```

! of routine PUSH_FAKE

				000C 00000 PUSH_FAKE:					
	53	00000000G	00	9E	00002	.WORD	Save R2,R3	: 1344	
	5E		08	C2	00009	MOVAB	OTSS\$L_CUR_LUN, R3	:	
	52		63	D0	0000C	SUBL2	#8, SP	: 1386	
		04	AE	9F	0000F	MOVL	OTSS\$L_CUR_LUN, LUN	: 1395	
04	AE	0121	8F	3C	00012	PUSHAB	PUSH	:	
		04	AE	9F	00018	MOVZWL	#289, 4(SP)	:	
00000000G	00		02	FB	0001B	PUSHAB	4(SP)	:	
	04		50	E8	00022	CALLS	#2, LIB\$GET_VM	: 1397	
	50		03	D0	00025	BLBS	GET_VM_RESULT, 1\$:	
				04	00028	MOVL	#3, R0	:	
	50	04	AE	D0	00029	RET		: 1403	
10	A0		52	B0	0002D	MOVL	PUSH, R0	:	
14	A0		C2	88	00031	MOVW	LUN, 16(R0)	: 1404	
00000000'	EF		60	0E	00035	BISB2	#2, 20(R0)	: 1408	
	63	78	8F	9A	0003C	INSQUE	(R0), OTSSQ_IO_ACTIVE	: 1414	
	50		01	D0	00040	MOVZBL	#120, OTSS\$C_CUR_LUN	: 1415	
			04	00043	MOVL	#1, R0	: 1416		
					RET			:	

; Routine Size: 68 bytes, Routine Base: _OTSS\$CODE + 005B


```

497 1417 1 ROUTINE PUSH_ACTIVE (LOGICAL_UNIT,      ! The new LUN
498 1418 1     RECURSIVE_IO                          ! True if really recursive I/O
499 1419 1     ) : CALL_CCB =
500 1420 1
501 1421 1 ++
502 1422 1 FUNCTIONAL DESCRIPTION:
503 1423 1
504 1424 1     Place the ISB, etc. of the currently active logical unit on the
505 1425 1     I/O Active queue so that another I/O statement may be started.
506 1426 1     The I/O statement to be started may be on the same or another
507 1427 1     logical unit as the one being interrupted. When the new I/O
508 1428 1     statement is complete the old one will be continued, so the I/O
509 1429 1     active queue has a first-in-first-out discipline.
510 1430 1
511 1431 1 CALLING SEQUENCE:
512 1432 1
513 1433 1     RESULT = CALL PUSH_ACTIVE (LUN, RECURSIVE_IO);
514 1434 1
515 1435 1 FORMAL PARAMETERS:
516 1436 1
517 1437 1     LOGICAL_UNIT.rl.v      The new LUN
518 1438 1     RECURSIVE_IO.rl.v     True if this LUN was already active
519 1439 1
520 1440 1 IMPLICIT INPUTS:
521 1441 1
522 1442 1     OTSS$AA_LUB_TAB
523 1443 1     OTSS$Q_IO_ACTIVE
524 1444 1     OTSS$L_CUR_LUN
525 1445 1
526 1446 1 IMPLICIT OUTPUTS:
527 1447 1
528 1448 1     OTSS$Q_IO_ACTIVE      Holds previous I/O on this LUN
529 1449 1
530 1450 1 SIDE EFFECTS:
531 1451 1
532 1452 1     Calls LIB$GET_VM to get virtual memory.
533 1453 1 --
534 1454 1
535 1455 2 BEGIN
536 1456 2
537 1457 2 EXTERNAL REGISTER
538 1458 2     CCB : REF BLOCK [, BYTE];
539 1459 2
540 1460 2 LOCAL
541 1461 2 ++
542 1462 2 Declare the pointer to the block to push.
543 1463 2 --
544 1464 2     PUSH : REF BLOCK [PUSH$K_LENGTH, BYTE] FIELD (PUSH_ITEM);
545 1465 2
546 1466 2 ++
547 1467 2 If there is no need to push anything, push a fake activation
548 1468 2 record to satisfy POP_ACTIVE.
549 1469 2 --
550 1470 2
551 1471 2     IF (.OTSS$L_CUR_LUN GTR LUB$K_LUN_MAX) THEN RETURN (PUSH_FAKE ());
552 1472 2
553 1473 2 !+

```

```

554 1474 2 | Check for this being an AST between the clearing of OTSS$V IOINPROG
555 1475 2 | and the setting of OTSS$L_CUR_LUN to LUB$K_LUN_MAX + 1. If it
556 1476 2 | is we cannot push the CCB, since, with RECURSIVE_IO clear,
557 1477 2 | OTSS$V_IOINPROG will be cleared before the call to POP_ACTIVE,
558 1478 2 | and we might try to pop into a deallocated CCB.
559 1479 2 |
560 1480 2 |
561 1481 2 | IF ((.OTSS$L_CUR_LUN EQL .LOGICAL_UNIT) AND ( NOT .RECURSIVE_IO)) THEN RETURN (PUSH_FAKE ());
562 1482 2 |
563 1483 2 | CCB = .OTSS$AA_LUB_TAB [.OTSS$L_CUR_LUN, 0];
564 1484 2 |
565 1485 2 | + If the queue is empty then the deallocation code has removed the LUB
566 1486 2 | from the LUB table but has not yet popped OTSS$L_CUR_LUN. Since
567 1487 2 | the deallocation code will finish its deallocation no matter what
568 1488 2 | we do here we need not push anything. If any I/O is tried to this
569 1489 2 | LUN it will create a new LUB. The recursive flag may be set
570 1490 2 | needlessly, but that will only cause a problem in languages which
571 1491 2 | do not support recursive I/O, and, actually, the higher I/O has not
572 1492 2 | quite finished yet, so that is OK.
573 1493 2 |
574 1494 2 |
575 1495 2 | IF (.CCB EQLA OTSS$AA_LUB_TAB [.OTSS$L_CUR_LUN, 0]) THEN RETURN (PUSH_FAKE ());
576 1496 2 |
577 1497 2 | +
578 1498 2 | The LUB is still allocated, do some consistency checks.
579 1499 2 | We cannot check OTSS$AA_CUR_LUB since we may be in an AST that
580 1500 2 | occurred after the update of OTSS$AA_CUR_LUB but before OTSS$L_CUR_LUN.
581 1501 2 |
582 1502 2 | CCB = .CCB + (.CCB - CCB [LUB$Q_QUEUE]);
583 1503 2 |
584 1504 2 | IF (.CCB [LUB$W_LUN] NEQ .OTSS$L_CUR_LUN) THEN LIB$STOP (OTSS_FATINTERR);
585 1505 2 |
586 1506 2 | +
587 1507 2 | Get virtual memory to hold the old ISB, etc.
588 1508 2 |
589 1509 2 | BEGIN
590 1510 2 |
591 1511 2 | LOCAL
592 1512 2 | GET_VM_RESULT;
593 1513 2 |
594 1514 2 | GET_VM_RESULT = LIB$GET_VM (%REF (PUSH$K_LENGTH), PUSH);
595 1515 2 |
596 1516 2 | IF ( NOT .GET_VM_RESULT) THEN RETURN (OTSS$K_PUSH_FAIL);
597 1517 2 |
598 1518 2 | END;
599 1519 2 | +
600 1520 2 | Make sure there is no RMS I/O active on the RAB.
601 1521 2 |
602 1522 2 |
603 1523 2 | IF (.RECURSIVE_IO) THEN $WAIT (RAB = .CCB);
604 1524 2 |
605 1525 2 | +
606 1526 2 | Copy the ISB and a few other things that need to be preserved
607 1527 2 | over recursive I/O into the block we just allocated.
608 1528 2 |
609 1529 2 | CH$MOVE (ISB$K_ISB_LEN, .CCB - ISB$K_ISB_LEN - LUB$K_LUB_LEN, PUSH [PUSH$X_ISB]);
610 1530 2 | PUSH [PUSH$V_PMT] = .CCB [RAB$V_PMT];
```



```

611 1531
612 1532
613 1533
614 1534
615 1535
616 1536
617 1537
618 1538
619 1539
620 1540
621 1541
622 1542
623 1543
624 1544
625 1545
626 1546
627 1547
628 1548
629 1549
630 1550
631 1551
632 1552
633 1553
634 1554
635 1555
636 1556
637 1557
638 1558
639 1559
640 1560
641 1561
642 1562
643 1563
644 1564

```

```

IF (.PUSH [PUSH$V_PMT])
THEN
  BEGIN
    CH$MOVE (.CCB [RAB$B_PSZ], .CCB [RAB$L_PBF], PUSH [PUSH$T_PROMPT]);
    PUSH [PUSH$B_PSZ] = .CCB [RAB$B_PSZ];
    END;

    PUSH [PUSH$B_TMO] = .CCB [RAB$B_TMO];
    PUSH [PUSH$L_STS] = .CCB [RAB$L_STS];
    PUSH [PUSH$L_STV] = .CCB [RAB$L_STV];
    PUSH [PUSH$V_IO_ACT] = .CCB [LUB$V_IO_ACTIVE];
    PUSH [PUSH$V_FAKE] = 0;

    + Record the logical unit number so that POP_ACTIVE knows where to
    + restore this item when it is popped.
    PUSH [PUSH$W_LUN] = .CCB [LUB$W_LUN];

    + Put this item on the I/O Active list.
    INSQUE (.PUSH, OTSS$Q_IO_ACTIVE);

    + That LUB is no longer the active one, mark it so.
    CCB [LUB$V_IO_ACTIVE] = 0;

    + We also set OTSS$L_CUR_LUN to LUB$K_LUN_MAX+1 to prevent an
    + AST from pushing that LUB again. An extra push before this point
    + does not cause any harm (only wastes a little time).
    OTSS$L_CUR_LUN = LUB$K_LUN_MAX + 1;
    RETURN (OTSS$K_PUSH_OK);
  END;

```

! of routine PUSH_ACTIVE

```

                                .EXTRN  SYSS$WAIT
                                00FC 00000 PUSH_ACTIVE:
                                .WORD
00000077 57 00000000G 00 9E 00002  MOVAB  Save R2,R3,R4,R5,R6,R7
                                5E 08 C2 00009  SUBL2  OTSS$L_CUR_LUN, R7
                                8F 67 D1 0000C  CMPL  #8, SP
                                1D 14 00013  BGTR  OTSS$L_CUR_LUN, #119
                                04 AC 67 D1 00015  CMPL  2$
                                13 08 AC E9 0001B  BNEQ  OTSS$L_CUR_LUN, LOGICAL_UNIT
                                50 67 D0 0001F 1$: BLBC  1$
                                50 00000000G0040 7E 00022  MOVL  RECURSIVE IO, 2$
                                5B 60 D0 0002A  MOVAQ  OTSS$L_CUR_LUN, R0
                                50 5B D1 0002D  MOVAB  OTSS$AX_LUB_TAB+64[R0], R0
                                86 AF 00 12 00030  CMPL  (R0), CCB
                                5B 05 12 0003D  CMPL  CCB, R0
                                5B 00 FB 00032 2$: BNEQ  3$
                                5B 04 00036  RET
                                5B 5B C3 00037 3$: SUBL3  CCB, CCB, R0
                                5B 5B A04B 9E 0003B  MOVAB  88(R0)[CCB], CCB

```

67	C6	AB	10	00	EC	00040	CMPV	#0, #16, -58(CCB), OTSS\$SL_CUR_LUN	1504
				0D	13	00046	BEQL	4\$	
			00000000G	00	8F	DD	00048	PUSHL	NOTSS\$FATINTERR
				04	01	FB	0004E	CALLS	#1, LIB\$STOP
				04	AE	9F	00055	PUSHAB	PUSH
			04	AE	8F	3C	00058	MOVZWL	#289, 4(SP)
				04	AE	9F	0005E	PUSHAB	4(SP)
			00000000G	00	02	FB	00061	CALLS	#2, LIB\$GET_VM
				04	50	E8	00068	BLBS	GET_VM_RESULT, 5\$
				50	03	DO	0006B	MOVL	#3, R0
					04	0006E	RET		
				09	08	AC	E9	0006F	5\$: RECURSIVE_IO, 6\$
					5B	DD	00073	PUSHL	CCB
			00000000G	00	01	FB	00075	CALLS	#1, SYS\$WAIT
				56	04	AE	DO	0007C	6\$: MOVL PUSH, R6
			FEE0	CB	00BC	8F	28	00080	MOV C3
				01	06	EF	00089	EXTZV	#188, -288(CCB), 101(R6)
				02	50	FO	0008F	INSV	#6, #1, 7(CCB), R0
				02	E1	00095	BBC	R0, #2, #1, 20(R6)	
				50	34	AB	9A	0009A	#2, 20(R6), 7\$
					50	28	0009E	MOVZBL	52(CCB), R0
					AB	90	000A4	MOV C3	R0, #48(CCB), 21(R6)
					1F	AB	90	000A9	7\$: MOVB 52(CCB), 18(R6)
					08	AB	7D	000AE	MOVB 31(CCB), 19(R6)
				01	01	EF	000B3	MOVG	8(CCB), 8(R6)
				00	50	FO	000B9	EXTZV	#1, #1, -4(CCB), R0
					02	8A	000BF	INSV	R0, #0, #1, 20(R6)
					AB	B0	000C3	BICB2	#2, 20(R6)
			00000000*	EF	66	0E	000C8	MOVW	-58(CCB), 16(R6)
			FC	AB	02	8A	000CF	INSQUE	(R6), OTSS\$Q_IO_ACTIVE
				67	8F	9A	000D3	BICB2	#2, -4(CCB)
				50	01	DO	000D7	MOVZBL	#120, OTSS\$SL_CUR_LUN
					04	000DA	MOVL	#1, R0	
							RET		1564

; Routine Size: 219 bytes, Routine Base: _OTSS\$CODE + 009F


```

: 646 1565 1 ROUTINE ALLOCATE (LOGICAL_UNIT) : CALL_CCB = ! Allocate LUB/ISB/RAB
: 647 1566 1
: 648 1567 1 ++
: 649 1568 1 FUNCTIONAL DESCRIPTION:
: 650 1569 1
: 651 1570 1 Allocate the LUB/ISB/RAB for this logical unit, watching out for
: 652 1571 1 ASTs which may do the allocation as we are running.
: 653 1572 1
: 654 1573 1 CALLING SEQUENCE:
: 655 1574 1
: 656 1575 1 CALL ALLOCATE (.LOGICAL_UNIT)
: 657 1576 1
: 658 1577 1 FORMAL PARAMETERS:
: 659 1578 1
: 660 1579 1 LOGICAL_UNIT.rl.v The logical unit number for this CCB
: 661 1580 1
: 662 1581 1 IMPLICIT INPUTS:
: 663 1582 1
: 664 1583 1 OTSS$AA_LUB_TAB
: 665 1584 1
: 666 1585 1 IMPLICIT OUTPUTS:
: 667 1586 1
: 668 1587 1 OTSS$AA_LUB_TAB
: 669 1588 1 CCB
: 670 1589 1
: 671 1590 1 SIDE EFFECTS:
: 672 1591 1
: 673 1592 1 Calls LIB$GET_VM to get virtual memory.
: 674 1593 1 May call LIB$FREE_VM to free that same virtual memory.
: 675 1594 1 --
: 676 1595 1
: 677 1596 2 BEGIN
: 678 1597 2
: 679 1598 2 EXTERNAL REGISTER
: 680 1599 2 CCB : REF BLOCK [, BYTE];
: 681 1600 2
: 682 1601 2 LOCAL
: 683 1602 2 INSQUE_ADDR, ! Address for INSQUE instruction
: 684 1603 2 REMQUE_ADDR, ! Address for REMQUE instruction
: 685 1604 2 CCB_ADDR; ! Address of the allocated CCB
: 686 1605 2
: 687 1606 2
: 688 1607 2 +
: 689 1608 2 Test the definitions of the LUB and ISB for consistency. This is
: 690 1609 2 purely a compile-time test; it generates no code.
: 691 1610 2
: 692 1611 2 TEST_LUB_ISB;
: 693 1612 2 +
: 694 1613 2 We must allocate. This case is a little complex since an AST may
: 695 1614 2 allocate the LUB. We handle this by preparing the LUB and then
: 696 1615 2 checking to see if an AST allocated one. If so, we deallocate ours.
: 697 1616 2
: 698 1617 2 BEGIN
: 699 1618 2
: 700 1619 2 LOCAL
: 701 1620 2 GET_VM_RESULT;
: 702 1621 3 GET_VM_RESULT = LIB$GET_VM (%REF (K_TOTAL_CCB_LEN), CCB_ADDR);

```

```

1622      IF ( NOT .GET_VM_RESULT) THEN RETURN (OTSS$K_PUSH_FAIL);
1623
1624      END;
1625
1626      + Clear the newly allocated LUN and RAB (but not ISB). Adjust the
1627      contents of the control block pointer (CCB) so that it points to
1628      the beginning of the RAB. (The ISB and LUB precede the RAB using
1629      negative offsets with respect to register CCB.)
1630      Set the unit number in the newly allocated LUB.
1631
1632      - CCB = .CCB_ADDR;
1633      CH$FILL (0, LUB$K_LUB_LEN + RAB$C_BLN, .CCB + ISB$K_ISB_LEN);
1634      CCB = .CCB + ISB$K_ISB_LEN + LUB$K_LUB_LEN;
1635      CCB [LUB$W_LUN] = .LOGICAL_UNIT;
1636
1637      + Initialize RAB to constants which never change.
1638      Block ID, block length, and bit to make $PUT do $UPDATE if
1639      record exists. Also truncate on sequential $PUT not at EOF.
1640      Note: TPT bit depends on FOP TRN bit being set in order to take effect.
1641      Set read-ahead, write-behind and locate mode for GETs.
1642
1643      - CCB [RAB$B_BID] = RAB$C_BID;
1644      CCB [RAB$B_BLN] = RAB$C_BLN;
1645      CCB [RAB$V_UIF] = 1;
1646      CCB [RAB$V_TPT] = 1;
1647      CCB [RAB$V_RAH] = 1;
1648      CCB [RAB$V_WBH] = 1;
1649      CCB [RAB$V_LOC] = 1;
1650
1651      + Set up LUB$A_BUDDY_PTR. If this CCB is not its own buddy, this
1652      field will be changed during open.
1653
1654      - CCB [LUB$A_BUDDY_PTR] = .CCB;
1655
1656      + See if an AST has allocated this LUB/RAB/ISB while we were preparing
1657      ours above. If so, we use the allocated one. If the LUB was
1658      allocated by an AST it cannot have I/O active, since the AST must
1659      complete any I/O it starts. In spite of this, it cannot be
1660      deallocated because we have OTSS$V_IOINPROG set for the LUN.
1661
1662      - INSQUE_ADDR = OTSS$AA_LUB_TAB [.LOGICAL_UNIT, 1];
1663
1664      IF ( NOT INSQUE (CCB [LUB$Q_QUEUE], ..INSQUE_ADDR))
1665      THEN
1666      BEGIN
1667
1668      + This CCB is not the first in the queue, which means that an AST
1669      has allocated one and put it in the queue before us. Remove ours
1670      and deallocate it. We will use the LUB previously on the queue.
1671
1672      - REMQUE_ADDR = OTSS$AA_LUB_TAB [.LOGICAL_UNIT, 1];
1673
1674      CASE (REMQUE (..REMQUE_ADDR, CCB)) FROM 0 TO 3 OF
1675      SET
1676
1677      [2] :
1678
```



```

760 1679
761 1680
762 1681
763 1682
764 1683
765 1684
766 1685
767 1686
768 1687
769 1688
770 1689
771 1690
772 1691
773 1692
774 1693
775 1694
776 1695
777 1696
778 1697
779 1698
780 1699
781 1700
782 1701
783 1702
784 1703
785 1704
786 1705
787 1706
788 1707
789 1708
790 1709
791 1710
792 1711
793 1712
794 1713
795 1714
796 1715
797 1716
798 1717
799 1718
800 1719
801 1720
802 1721
803 1722
804 1723
805 1724
806 1725
807 1726
808 1727
809 1728
810 1729
811 1730

+
- Somebody removed the other entry. This should never happen.
  LIB$STOP (OTSS$_FATINTERR);
[3] :
+
- The queue was empty. This is unreasonable because OTSS$V_IOINPROG is set.
  LIB$STOP (OTSS$_FATINTERR);
[0] :
+
- All is well. We can now free the CCB we just removed.
  It had better be the one we allocated.
  IF ((.CCB + (.CCB - CCB [LUB$Q_QUEUE])) NEQA (.CCB_ADDR + ISB$K_ISB_LEN + LUB$K_LUB_LEN))
  THEN
    LIB$STOP (OTSS$_FATINTERR);
[INRANGE, OTRANGE] :
+
- This should never happen; the only possible values from the REMQUE
  function are 0, 2 and 3.
  LIB$STOP (OTSS$_FATINTERR);
  TES;
+
- Now free the LUB we allocated.
  BEGIN
  LOCAL
    FREE_VM_STATUS;
    FREE_VM_STATUS = LIB$FREE_VM (%REF (K_TOTAL_CCB_LEN), CCB_ADDR);
    IF ( NOT .FREE_VM_STATUS) THEN LIB$STOP (OTSS$_FATINTERR);
  END;
+
- Now fetch the CCB address. It must still be there because of
  OTSS$V_IOINPROG.
  CCB = .OTSS$AA_LUB_TAB [.LOGICAL_UNIT, 0];
  CCB = .CCB + (.CCB - CCB [LUB$Q_QUEUE]);
  END;
  RETURN (OTSS$K_PUSH_OK);
  END;
! of routine ALLOCATE
```

01FC 00000 ALLOCATE:

			58	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8	1565	
			57	00000000G	8F	D0	00009	MOVAB	LIB\$STOP, R8		
			56	00000000G	00	9E	00010	MOVL	#OTSS\$FATINTERR, R7		
			5E		08	C2	00017	MOVAB	OTSS\$AA_LUB_TAB+68, R6		
				04	AE	9F	0001A	SUBL2	#8, SP		
	04			0164	8F	3C	0001D	PUSHAB	CCB_ADDR	1621	
				04	AE	9F	00023	MOVZWL	#356, 4(SP)		
								PUSHAB	4(SP)		
	00000000G		00		02	FB	00026	CALLS	#2, LIB\$GET_VM		
			C4		50	E8	0002D	BLBS	GET_VM_RESULT, 1\$	1623	
			50		03	D0	00030	MOVL	#3, -R0		
						04	00033	RET			
00A8	8F	00	5B		04	AE	D0	00034	1\$: MOVL	CCB_ADDR, CCB	1633
			6E			00	2C	00038	MOVCS	#0, -(SP), #0, #168, 188(CCB)	1634
				00BC	CB		0003F				
			5B	0120	CB	9E	00042	MOVAB	288(R11), CCB	1635	
	C6		AB	04	AC	B0	00047	MOVW	LOGICAL_UNIT, -58(CCB)	1636	
			6B	4401	8F	B0	0004C	MOVW	#17409, -(CCB)	1644	
			50	04	AB	9E	00051	MOVAB	4(CCB), R0	1646	
			60	00010612	8F	C8	00055	BISL2	#67090, (R0)	1650	
	B8		AB		5B	D0	0005C	MOVL	CCB, -72(CCB)	1655	
			50	04	AC	D0	00060	MOVL	LOGICAL_UNIT, R0	1663	
			50		6640	7E	00064	MOVAQ	OTSS\$AA_LUB_TAB+68[R0], INSQUE_ADDR		
	00		B0	AB	0E	00068	INSQUE	-88(CCB), @0(INSQUE_ADDR)		1665	
					6C	13	0006D	BEQL	7\$		
			50	04	AC	D0	0006F	MOVL	LOGICAL_UNIT, R0	1673	
			51		6640	7E	00073	MOVAQ	OTSS\$AA_LUB_TAB+68[R0], REMQUE_ADDR		
			5B	00	B1	0F	00077	REMQUE	@0(REMQUE_ADDR), CCB	1675	
					50	DC	0007B	MOVPSL	R0		
50	50		02		01	EF	0007D	EXTZV	#1, #2, R0, R0		
	03		00		50	CF	00082	CASEL	R0, #0, #3		
0021	0021	0021			000A		00086	2\$: .WORD	3\$-2\$,-		
									4\$-2\$,-		
									4\$-2\$,-		
									4\$-2\$,-		
									4\$-2\$		
									4\$	1688	
50			5B		17	11	0008E	BRB		1696	
			51	58	5B	C3	00090	SUBL3	CCB, CCB, R0		
50	04		AE	00000120	A04B	9E	00094	MOVAB	88(R0)[CCB], R1		
			50		8F	C1	00099	ADDL3	#288, CCB_ADDR, R0		
					51	D1	000A2	CMPL	R1, R0		
					05	13	000A5	BEQL	5\$		
					57	DD	000A7	4\$: PUSHL	R7	1698	
			68		01	FB	000A9	CALLS	#1, LIB\$STOP		
				04	AE	9F	000AC	5\$: PUSHAB	CCB_ADDR	1716	
	04			0164	8F	3C	000AF	MOVZWL	#356, 4(SP)		
				04	AE	9F	000B5	PUSHAB	4(SP)		
	00000000G		00		02	FB	000B8	CALLS	#2, LIB\$FREE_VM		
			05		50	E8	000BF	BLBS	FREE_VM_STATOS, 6\$	1718	
					57	DD	000C2	PUSHL	R7		
			68		01	FB	000C4	CALLS	#1, LIB\$STOP		
			50		04	AC	D0	000C7	6\$: MOVL	LOGICAL_UNIT, R0	1725
				FC	A640	7F	000CB	PUSHAQ	OTSS\$AA_LUB_TAB+64[R0]		
			5B		9E	D0	000CF	MOVL	@(SP)+, CCB		
			5B		5B	C3	000D2	SUBL3	CCB, CCB, R0	1726	
50			5B	58	A04B	9E	000D6	MOVAB	88(R0)[CCB], CCB		
			50		01	D0	000DB	7\$: MOVL	#1, R0	1729	

OTSS\$CCB
1-057

K 7
16-Sep-1984 01:22:30
14-Sep-1984 12:39:38

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]OTSCCB.B32;1

Page 22
(7)

04 000DE

RET

; 1730

; Routine Size: 223 bytes, Routine Base: _OTSS\$CODE + 017A

; 812 1731 1

```
814 1732 1 GLOBAL ROUTINE OTSS$PUSH_CCB (          ! Get a CCB, pushing old
815 1733 1      LOGICAL UNIT                      ! Logical unit for this CCB
816 1734 1 )      : JSB_CB_POSH =
817 1735 1
818 1736 1 ++
819 1737 1 FUNCTIONAL DESCRIPTION:
820 1738 1
821 1739 1      Load register CCB with a pointer to the LUB/ISB/RAB for this
822 1740 1      logical unit.  If no LUB has been allocated, allocate one.
823 1741 1      If there is already I/O active push down the old ISB, etc.
824 1742 1      POP_ACTIVE will restore it.  We already know that this LUN is
825 1743 1      not in use by FORTRAN.
826 1744 1
827 1745 1 CALLING SEQUENCE:
828 1746 1
829 1747 1      CALL OTSS$PUSH_CCB (logical_unit.rl.v)
830 1748 1
831 1749 1 FORMAL PARAMETERS:
832 1750 1
833 1751 1      logical_unit.rl.v      Logical unit - identifies CCB
834 1752 1
835 1753 1 IMPLICIT INPUTS:
836 1754 1
837 1755 1      OTSS$V_CCB_INIT
838 1756 1      OTSS$AA_LUB_TAB
839 1757 1      OTSS$Q_IO_ACTIVE
840 1758 1
841 1759 1 IMPLICIT OUTPUTS:
842 1760 1
843 1761 1      CCB                      Set to adr. of allocated LUB/ISB/RAB
844 1762 1      OTSS$Q_IO_ACTIVE        Holds previous I/O on this LUN
845 1763 1      OTSS$AA_LUB_TAB        Set to adr. of allocated LUB/ISB/RAB
846 1764 1                        for logical_unit
847 1765 1      LUB$W_LUN                Set to logical_unit
848 1766 1      LUB$V_IO_ACTIVE          Set to indicate active I/O
849 1767 1      OTSS$V_CCB_INIT          Always set to 1.
850 1768 1
851 1769 1 SIDE EFFECTS:
852 1770 1
853 1771 1      May call LIB$GET_VM to get virtual memory.
854 1772 1      In unusual cases, may call LIB$FREE_VM to free virtual memory.
855 1773 1      The first time entered, calls INITIALIZE, which disables ASTs.
856 1774 1 --
857 1775 1
858 1776 2 BEGIN
859 1777 2
860 1778 2 EXTERNAL REGISTER
861 1779 2      CCB : REF BLOCK [, BYTE];
862 1780 2
863 1781 2 LOCAL
864 1782 2      RECURSIVE_IO;          ! =1 if we are doing recursive I/O
865 1783 2
866 1784 2 ++
867 1785 2      If this is the first entry, call INITIALIZE to set up OWN storage.
868 1786 2      Note that PUSH_CCB must be entered before POP_CCB, so this is the
869 1787 2      first reference to this data base, except for FORTRAN, which is checked
870 1788 2      for in INITIALIZE.
```



```

871 1789 2 !-
872 1790 2
873 1791 2 IF ( NOT .OTSS$V_CCB_INIT) THEN INITIALIZE ();
874 1792 2
875 1793 2
876 1794 2 + Count the level counter. This must be done before the OTSS$V_IOINPROG
877 1795 2 bit is set, otherwise an AST could find the OTSS$V_IOINPROG bit set but
878 1796 2 level counter -1, which would mean that the PUSH and POP routines
879 1797 2 would not be called and OTSS$V_IOINPROG would get cleared by the AST.
880 1798 2
881 1799 2 OTSS$L_LVL_CTR = .OTSS$L_LVL_CTR + 1;
882 1800 2
883 1801 2 + Mark that this LUN has I/O active so that its LUB (if it has one yet)
884 1802 2 will not be deallocated. If it was already active, remember that.
885 1803 2
886 1804 2 RECURSIVE_IO = (TESTBITSS (OTSS$V_IOINPROG [.LOGICAL_UNIT - LUB$K_ILUN_MIN]));
887 1805 2
888 1806 2 + If I/O is currently active, push the presently active unit.
889 1807 2
890 1808 2
891 1809 2 IF (.OTSS$L_LVL_CTR NEQ 0)
892 1810 2 THEN
893 1811 2 BEGIN
894 1812 2
895 1813 2 LOCAL
896 1814 2 PUSH_RESULT;
897 1815 2
898 1816 2 PUSH_RESULT = PUSH_ACTIVE (.LOGICAL_UNIT, .RECURSIVE_IO);
899 1817 2
900 1818 2 IF (.PUSH_RESULT NEQ OTSS$K_PUSH_OK) THEN RETURN (.PUSH_RESULT);
901 1819 2
902 1820 2 END;
903 1821 2
904 1822 2 +
905 1823 2 Allocate the LUB/ISB/RAB if necessary. If an AST allocates it we
906 1824 2 must release ours. Note that, because OTSS$V_IOINPROG is set, if an
907 1825 2 AST allocates the LUB it will not be deallocated.
908 1826 2
909 1827 2 CCB = .OTSS$AA_LUB_TAB [.LOGICAL_UNIT, 0];
910 1828 2
911 1829 2 IF (.CCB NEQ OTSS$AA_LUB_TAB [.LOGICAL_UNIT, 0])
912 1830 2 THEN
913 1831 2 BEGIN
914 1832 2
915 1833 2 + The CCB is already allocated. Adjust register CCB to point to it.
916 1834 2
917 1835 2 CCB = .CCB + (.CCB - CCB [LUB$Q_QUEUE]);
918 1836 2 END
919 1837 2 ELSE
920 1838 2 BEGIN
921 1839 2
922 1840 2 LOCAL
923 1841 2 ALLOCATE_RESULT;
924 1842 2
925 1843 2 ALLOCATE_RESULT = ALLOCATE (.LOGICAL_UNIT);
926 1844 2
927 1845 2 IF (.ALLOCATE_RESULT NEQ OTSS$K_PUSH_OK) THEN RETURN (.ALLOCATE_RESULT);
```

```

: 928      1846      3
: 929      1847      3
: 930      1848      3
: 931      1849      3
: 932      1850      3
: 933      1851      3
: 934      1852      3
: 935      1853      3
: 936      1854      3
: 937      1855      3
: 938      1856      3
: 939      1857      3
: 940      1858      3
: 941      1859      3
: 942      1860      3
: 943      1861      3
: 944      1862      3
: 945      1863      3
: 946      1864      3
: 947      1865      3
: 948      1866      3
: 949      1867      3
: 950      1868      3
: 951      1869      3
: 952      1870      3
: 953      1871      3
: 954      1872      3
: 955      1873      3
: 956      1874      3
: 957      1875      3
: 958      1876      3
: 959      1877      3
: 960      1878      3
: 961      1879      3
: 962      1880      3
: 963      1881      3
: 964      1882      3
: 965      1883      3
: 966      1884      3
: 967      1885      1

      END;

+ Set OTSS$L_CUR_LUN to be the current logical unit number. This is
- the cell that controls pushing.

      OTSS$L_CUR_LUN = .LOGICAL_UNIT;

+ Mark this LUB as being the active one, and, if it is participating
- in recursive I/O, mark that, too.

      CCB [LUB$V_IO_ACTIVE] = 1;
      CCB [ISB$V_RECURSIVE] = .RECURSIVE_IO;

+ Set OTSS$A_CUR_LUB to point to the new current LUB.
-

      OTSS$A_CUR_LUB = .CCB;

+ Initialize the STTM_STAT field of the ISB. We clear these bits so
- that the initialization routines at UDF and REC levels can set them
  if necessary (unusual) or do nothing to have them cleared.

      CCB [ISB$V_P_FORM_CH] = 0;
      CCB [ISB$V_DOLLAR] = 0;
      CCB [ISB$V_USER_ELEM] = 0;
      CCB [ISB$V_SLASH] = 0;
      CCB [ISB$V_LAST_REC] = 0;
      CCB [ISB$V_DE_ENCODE] = 0;
      CCB [ISB$V_LIS_HEAP] = 0;

+ When we set OTSS$V_IOINPROG we tested it to see if I/O was already active
- on this LUN. If it was we must return this information to our
  caller because some languages do not permit recursive I/O.

      IF (.RECURSIVE_IO) THEN RETURN (OTSS$K_PUSH_ACT);

      RETURN (OTSS$K_PUSH_OK);
      END;

! End of routine OTSS$PUSH_CCB
```

```

52 DD 00000 OTSS$PUSH_CCB::
      5E      04 C2 00002      PUSH R2      1732
      FD94    52 DD 00005      SUBL2 #4, SP
      CF      00 E8 00007      PUSH R2
      00000000G 00 FB 0000E      BLBS OTSS$V_CCB_INIT, 1$      1791
      00000000G 00 D6 00013 1$:      CALLS #0, INITIACIZE
      52      08 C1 00019      INCL OTSS$L_LVL_CTR      1799
      6E      50 D4 0001D      ADDL3 #8, LOGICAL_UNIT, R2      1804
      02 00000000G 00      52 E3 0001F      CLRL R0
      04 AE      50 D6 00027      BBS R2, OTSS$V_IOINPROG, 2$
      50 D0 00029 2$:      INCL R0
      MOV L R0, RECURSIVE_IO
```


OTSS\$CCB
1-057

B 8
16-Sep-1984 01:22:30
14-Sep-1984 12:39:38

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]OTSCCB.B32;1

Page 26
(8)

		00000000G	00	D5	0002D	TSTL	OTSS\$L_LVL_CTR	:	1809	
			10	13	00033	BEQL	3\$:		
		04	AE	DD	00035	PUSHL	RECURSIVE_IO	:	1816	
		04	AE	DD	00038	PUSHL	LOGICAL_UNIT	:		
FE06	CF		02	FB	00038	CALLS	#2, PUSH_ACTIVE	:		
	01		50	D1	00040	CMPL	PUSH_RESULT, #1	:	1818	
			52	12	00043	BNEQ	7\$:		
	50	00000000G	042	7E	00045	3\$: MOVAQ	OTSS\$AA_LUB_TAB[R2], R0	:	1827	
	5B		60	D0	0004D	MOVL	(R0), CCB	:		
	50		5B	D1	00050	CMPL	CCB, R0	:	1829	
			0B	13	00053	BEQL	4\$:		
50	5B		5B	C3	00055	SUBL3	CCB, CCB, R0	:	1835	
	5B	58	A04B	9E	00059	MOVAB	88(R0)[CCB], CCB	:		
			0C	11	0005E	BRB	5\$:	1829	
			6E	DD	00060	4\$: PUSHL	LOGICAL_UNIT	:	1843	
FEBA	CF		01	FB	00062	CALLS	#1, ALLOCATE	:		
	01		50	D1	00067	CMPL	ALLOCATE_RESULT, #1	:	1845	
			2B	12	0006A	BNEQ	7\$:		
	00000000G	00	6E	D0	0006C	5\$: MOVL	LOGICAL_UNIT, OTSS\$L_CUR_LUN	:	1853	
	FC	AB	02	88	00073	BISB2	#2, -4(CCB)	:	1858	
		50	96	AB	9E	00077	MOVAB	-106(CCB), R0	:	1859
01	A0	01	04	AE	F0	0007B	INSV	RECURSIVE_IO, #0, #1, 1(R0)	:	
		00000000G	00	5B	D0	00082	MOVL	CCB, OTSS\$A_CUR_LUB	:	1863
			60	94	00089	CLRB	(R0)	:	1875	
			04	AE	E9	0008B	BLBC	RECURSIVE_IO, 6\$:	1882
	05		02	D0	0008F	MOVL	#2, R0	:		
	50		03	11	00092	BRB	7\$:		
			01	D0	00094	6\$: MOVL	#1, R0	:	1884	
	5E		08	C0	00097	7\$: ADDL2	#8, SP	:	1885	
			04	BA	0009A	POPR	#^M<R2>	:		
			05	0009C	RSB			:		

; Routine Size: 157 bytes, Routine Base: _OTSS\$CODE + 0259

; 968 1886 1

```
1887 1 ROUTINE DEALLOCATE : CALL_CCB NOVALUE = ! Deallocate LUB/ISB/RAB
1888 1
1889 1 ++
1890 1 FUNCTIONAL DESCRIPTION:
1891 1
1892 1 Deallocate the LUB/ISB/RAB for this logical unit, including
1893 1 the allocated structures attached to it. Also, deallocate the
1894 1 LUN.
1895 1
1896 1 CALLING SEQUENCE:
1897 1
1898 1 CALL DEALLOCATE ( )
1899 1
1900 1 FORMAL PARAMETERS:
1901 1
1902 1 NONE
1903 1
1904 1 IMPLICIT INPUTS:
1905 1
1906 1 OTSS$AA_LUB_TAB
1907 1 CCB
1908 1
1909 1 IMPLICIT OUTPUTS:
1910 1
1911 1 OTSS$AA_LUB_TAB
1912 1 CCB
1913 1
1914 1 SIDE EFFECTS:
1915 1
1916 1 Calls LIB$FREE_VM to free virtual memory.
1917 1 --
1918 1
1919 2 BEGIN
1920 2
1921 2 EXTERNAL REGISTER
1922 2 CCB : REF BLOCK [, BYTE];
1923 2
1924 2 LOCAL
1925 2 REMQUE_ADDR, ! Address for REMQUE instruction
1926 2 CCB_ADDR : REF BLOCK [0, BYTE],
1927 2 BUDDY_CCB : REF BLOCK [0, BYTE],
1928 2 LUN;
1929 2
1930 2 +
1931 2 We now deallocate the LUB/ISB/RAB. An AST will not deallocate under
1932 2 us because it will find OTSS$V_IOINPROG set at PUSH time, and will
1933 2 therefore set ISB$V_RECURSIVE so as not to clear OTSS$V_IOINPROG at POP
1934 2 time or deallocate the LUB.
1935 2 -
1936 2 REMQUE_ADDR = OTSS$AA_LUB_TAB [.CCB [LUB$W_LUN], 0];
1937 2
1938 2 CASE (REMQUE (..REMQUE_ADDR, CCB_ADDR)) FROM 0 TO 3 OF
1939 2 SET
1940 2 [0, 3] :
1941 2
1942 2 +
1943 2 ! Zero means that there was more than one entry in the queue.
```



```
1027 1944 2 | This implies that we have done a CLOSE in an AST which went off after
1028 1945 2 | the INSQUE but before the compensating REMQUE in PUSH_CCB.
1029 1946 2 | This should never happen because ISB$V_RECURSIVE
1030 1947 2 | will be set in this case.
1031 1948 2 |
1032 1949 2 | Three implies that there is nothing in the queue.
1033 1950 2 | This means that an AST deallocated the LUB, which should not happen
1034 1951 2 | because of the ISB$V_RECURSIVE test.
1035 1952 2 |
1036 1953 2 |     LIB$STOP (OTSS$_FATINTERR);
1037 1954 2 |
1038 1955 2 |     [2] :
1039 1956 2 |
1040 1957 2 | + The queue is now empty. This is correct. We can now free the LUB.
1041 1958 2 | Note that PUSH_CCB will allocate a new LUB if an AST goes off to it
1042 1959 2 | here, and will carefully not push the LUB we are deallocating.
1043 1960 2 | First perform a consistency check.
1044 1961 2 |
1045 1962 2 |
1046 1963 2 |     IF (.CCB_ADDR + (.CCB_ADDR - CCB_ADDR [LUB$Q_QUEUE]) NEQA .CCB) THEN LIB$STOP (OTSS$_FATINTERR);
1047 1964 2 |
1048 1965 2 |     [INRANGE, OTRANGE] :
1049 1966 2 |
1050 1967 2 | + This should never happen. The only possible values of REMQUE are
1051 1968 2 | 0, 2 and 3.
1052 1969 2 |
1053 1970 2 |     LIB$STOP (OTSS$_FATINTERR);
1054 1971 2 |     TES;
1055 1972 2 |
1056 1973 2 | + Since the LUB/ISB/RAB can no longer be used, clear its OTSS$V_IOINPROG
1057 1974 2 | bit. An AST after this point will not indicate recursive I/O.
1058 1975 2 |
1059 1976 2 |
1060 1977 2 |
1061 1978 2 |     IF (TESTBITCC (OTSS$V_IOINPROG [.CCB [LUB$W_LUN] - LUB$K_ILUN_MIN])) THEN LIB$STOP (OTSS$_FATINTERR);
1062 1979 2 |
1063 1980 2 | + Clear this LUN's buddy's buddy pointer, which points to us.
1064 1981 2 |
1065 1982 2 |     BUDDY_CCB = .CCB [LUB$A_BUDDY_PTR];
1066 1983 2 |
1067 1984 2 |     IF (.BUDDY_CCB NEQA 0) THEN BUDDY_CCB [LUB$A_BUDDY_PTR] = 0;
1068 1985 2 |
1069 1986 2 |
1070 1987 2 | + Free the record buffer if we allocated it.
1071 1988 2 |
1072 1989 2 |
1073 1990 2 |     IF (( NOT .CCB [LUB$V_USER_RBUF]) AND (.CCB [LUB$A_UBF] NEQA 0))
1074 1991 2 |     THEN
1075 1992 2 |     BEGIN
1076 1993 2 |         LOCAL
1077 1994 2 |         FREE_VM_STATUS;
1078 1995 2 |
1079 1996 2 |         FREE_VM_STATUS = LIB$FREE_VM (%REF (.CCB [LUB$W_RBUF_SIZE]), CCB [LUB$A_UBF]);
1080 1997 2 |
1081 1998 2 |         IF ( NOT .FREE_VM_STATUS) THEN LIB$STOP (OTSS$_FATINTERR);
1082 1999 2 |
1083 2000 2 |
```

```

: 1084      2001      3
: 1085      2002      3      END;
: 1086      2003      3
: 1087      2004      3      !+
: 1088      2005      3      !- Free the file name string, if it is allocated.
: 1089      2006      3
: 1090      2007      3
: 1091      2008      3      IF (.CCB [LUB$V_VIRT_RSN])
: 1092      2009      3      THEN
: 1093      2010      3      BEGIN
: 1094      2011      3
: 1095      2012      3      LOCAL
: 1096      2013      3      FREE_VM_STATUS;
: 1097      2014      3
: 1098      2015      3      FREE_VM_STATUS = LIB$FREE_VM (%REF (.CCB [LUB$B_RSL]), CCB [LUB$A_RSN]);
: 1099      2016      3
: 1100      2017      3      IF ( NOT .FREE_VM_STATUS) THEN LIB$STOP (OTSS$_FATINTERR);
: 1101      2018      3
: 1102      2019      3      CCB [LUB$V_VIRT_RSN] = 0;
: 1103      2020      3      END;
: 1104      2021      3
: 1105      2022      3      !+
: 1106      2023      3      !- Free the prompt buffer, if there is one.
: 1107      2024      3
: 1108      2025      3      BEGIN
: 1109      2026      3
: 1110      2027      3      LOCAL
: 1111      2028      3      FREE_VM_STATUS;
: 1112      2029      3
: 1113      2030      3      IF ((.CCB [RAB$L_PBF] NEQA 0) AND (.CCB [RAB$V_PMT]))
: 1114      2031      3      THEN
: 1115      2032      3      BEGIN
: 1116      2033      3      FREE_VM_STATUS = LIB$FREE_VM (%REF (LUB$K_PBUF_SIZ), CCB [RAB$L_PBF]);
: 1117      2034      3
: 1118      2035      3      IF ( NOT .FREE_VM_STATUS) THEN LIB$STOP (OTSS$_FATINTERR);
: 1119      2036      3
: 1120      2037      3      END;
: 1121      2038      3
: 1122      2039      3      END;
: 1123      2040      3      !+
: 1124      2041      3      !- Remember the logical unit number, since we will need it in a minute.
: 1125      2042      3
: 1126      2043      3      LUN = .CCB [LUB$W_LUN];
: 1127      2044      3      !+
: 1128      2045      3      !- Now, at last, we can free the CCB itself.
: 1129      2046      3
: 1130      2047      3      BEGIN
: 1131      2048      3
: 1132      2049      3      LOCAL
: 1133      2050      3      FREE_VM_STATUS;
: 1134      2051      3
: 1135      2052      3      FREE_VM_STATUS = LIB$FREE_VM (%REF (K_TOTAL_CCB_LEN), %REF (.CCB - ISB$K_ISB_LEN - LUB$K_LUB_LEN));
: 1136      2053      3
: 1137      2054      3      IF ( NOT .FREE_VM_STATUS) THEN LIB$STOP (OTSS$_FATINTERR);
: 1138      2055      3
: 1139      2056      3      END;
: 1140      2057      3      !+

```



```

: 1141 2058 2 | Since the CCB points to deallocated storage, clear register CCB so
: 1142 2059 2 | that, if anybody refers to it, we will get an access violation.
: 1143 2060 2 |
: 1144 2061 2 | CCB = 0;
: 1145 2062 2 |
: 1146 2063 2 | +
: 1147 2064 2 | If the user's program is still running (i.e., if we are not in the
: 1148 2065 2 | exit handler) then the user must have done an explicit CLOSE to cause
: 1149 2066 2 | this LUB to be deallocated. In that case we must clear the LUN
: 1150 2067 2 | allocation so he can do another OPEN on this same logical unit.
: 1151 2068 2 | Note that LUNs less than zero do not have allocation bits since they
: 1152 2069 2 | cannot be opened explicitly by the user.
: 1153 2070 2 |
: 1154 2071 2 | IF (.LUN GEQ 0)
: 1155 2072 2 | THEN
: 1156 2073 2 |
: 1157 2074 2 | IF ( NOT OTSS$FREE_LUN (LUN)) THEN LIB$STOP (OTSS_FATINTERR);
: 1158 2075 2 |
: 1159 2076 2 | RETURN;
: 1160 2077 1 | END;

```

! of routine DEALLOCATE

```

                                003C 00000 DEALLOCATE:
                                .WORD
55 00000000G 00 9E 00002      MOVAB  Save R2,R3,R4,R5
54 00000000G 00 9E 00009      MOVAB  LIB$FREE_VM, R5
53 00000000G 8F D0 00010      MOVL   LIB$STOP, R4
5E          0C C2 00017      MOVL   #OTSS_FATINTERR, R3
50          C6 AB 32 0001A     CVTWL  #12, SP
51 00000000G0040 7E 0001E     MOVAQ  -58(CCB), R0
52          00 B1 0F 00026     REMQUE OTSS$AA LUB TAB+64[R0], REMQUE_ADDR
          50 DC 0002A      REMQUE  @0(REMQUE_ADDR), CCB_ADDR
          01 EF 0002C      MOVPSL  R0
          50 CF 00031      EXTZV   #1, #2, R0, R0
          00          CASEL   R0, #0, #3
0018          0018 00035 1$:   .WORD  3$-1$, -
                                3$-1$, -
                                2$-1$, -
                                3$-1$
                                3$
                                1953
          50          52 0E 11 0003D      BRB   3$
          50          52 C3 0003F 2$:     SUBL3  CCB_ADDR, CCB_ADDR, R0
          5B          50 9E 00043      MOVAB  88(R0)[CCB_ADDR], R0
          5B          50 D1 00048      CMPL   R0, CCB
          5B          50 13 0004B      BEQL   4$
          5B          53 DD 0004D 3$:     PUSHL  R3
          64          01 FB 0004F      CALLS  #1, LIB$STOP
          50          C6 AB 32 00052 4$:     CVTWL  -58(CCB), R0
          50          08 C0 00056      ADDL2  #8, R0
          05 00000000G 00 50 E4 00059      BBSC   R0, OTSS$V_IOINPROG, 5$
          64          53 DD 00061      PUSHL  R3
          50          01 FB 00063      CALLS  #1, LIB$STOP
          50          B8 AB D0 00066 5$:     MOVL   -72(CCB), BUDDY_CCB
          50          03 13 0006A      BEQL   6$
          B8          A0 D4 0006C      CLRL   -72(BUDDY_CCB)
          FF          AB 95 0006F 6$:     TSTB   -1(CCB)
                                1991

```

			9C	1B	19	00072	BLSS	7\$		
				AB	D5	00074	TSTL	-100(CCB)		
				16	13	00077	BEQL	7\$		
			9C	AB	9F	00079	PUSHAB	-100(CCB)	1998	
	08	AE	D2	AB	3C	0007C	MOVZWL	-46(CCB), 8(SP)		
			08	AE	9F	00081	PUSHAB	8(SP)		
	65			02	FB	00084	CALLS	#2, LIB\$FREE_VM		
	05			50	E8	00087	BLBS	FREE_VM_STATOS, 7\$	2000	
				53	DD	0008A	PUSHL	R3		
	64			01	FB	0008C	CALLS	#1, LIB\$STOP		
	1A		FE	AB	E9	0008F	BLBC	-2(CCB), 9\$	2008	
			F8	AB	9F	00093	PUSHAB	-8(CCB)	2015	
	08	AE	F7	AB	9A	00096	MOVZBL	-9(CCB), 8(SP)		
			08	AE	9F	0009B	PUSHAB	8(SP)		
	65			02	FB	0009E	CALLS	#2, LIB\$FREE_VM		
	05			50	E8	000A1	BLBS	FREE_VM_STATOS, 8\$	2017	
				53	DD	000A4	PUSHL	R3		
	64			01	FB	000A6	CALLS	#1, LIB\$STOP		
	FE	AB		01	8A	000A9	BICB2	#1, -2(CCB)	2019	
			30	AB	D5	000AD	TSTL	48(CCB)	2030	
				1B	13	000B0	BEQL	10\$		
16	07	AB		06	E1	000B2	BBC	#6, 7(CCB), 10\$		
			30	AB	9F	000B7	PUSHAB	48(CCB)	2033	
	08	AE	50	8F	9A	000BA	MOVZBL	#80, 8(SP)		
			08	AE	9F	000BF	PUSHAB	8(SP)		
	65			02	FB	000C2	CALLS	#2, LIB\$FREE_VM		
	05			50	E8	000C5	BLBS	FREE_VM_STATOS, 10\$	2035	
				53	DD	000C8	PUSHL	R3		
	64			01	FB	000CA	CALLS	#1, LIB\$STOP		
	08	AE	C6	AB	32	000CD	CVTL	-58(CCB), LUN	2043	
	04	AE	FEE0	CB	9E	000D2	MOVAB	-288(R11), 4(SP)	2052	
			04	AE	9F	000D8	PUSHAB	4(SP)		
	04	AE	0164	8F	3C	000DB	MOVZWL	#356, 4(SP)		
			04	AE	9F	000E1	PUSHAB	4(SP)		
	65			02	FB	000E4	CALLS	#2, LIB\$FREE_VM		
	05			50	E8	000E7	BLBS	FREE_VM_STATOS, 11\$	2054	
				53	DD	000EA	PUSHL	R3		
	64			01	FB	000EC	CALLS	#1, LIB\$STOP		
				5B	D4	000EF	CLRL	CCB	2061	
			08	AE	D5	000F1	TSTL	LUN	2071	
				12	19	000F4	BLSS	12\$		
			08	AE	9F	000F6	PUSHAB	LUN	2074	
00000000G	00			01	FB	000F9	CALLS	#1, OTSS\$FREE_LUN		
	05			50	E8	00100	BLBS	R0, 12\$		
				53	DD	00103	PUSHL	R3		
	64			01	FB	00105	CALLS	#1, LIB\$STOP		
				04	00108	12\$:	RET		2077	

; Routine Size: 265 bytes, Routine Base: _OTSS\$CODE + 02F6


```

: 1162 2078 1 ROUTINE POP_ACTIVE : CALL_CCB NOVALUE =      ! Pop old active unit
: 1163 2079 1
: 1164 2080 1 ++
: 1165 2081 1 FUNCTIONAL DESCRIPTION:
: 1166 2082 1
: 1167 2083 1     Restore the status of an interrupted I/O statement using the
: 1168 2084 1     information saved when the statement was interrupted. All of
: 1169 2085 1     the ISB is restored, and a few other things. In some unusual
: 1170 2086 1     cases there is no CCB to restore to, so only OTSS$L_CUR_LUN is
: 1171 2087 1     restored.
: 1172 2088 1
: 1173 2089 1 CALLING SEQUENCE:
: 1174 2090 1
: 1175 2091 1     CALL POP_ACTIVE ()
: 1176 2092 1
: 1177 2093 1 FORMAL PARAMETERS:
: 1178 2094 1
: 1179 2095 1     NONE
: 1180 2096 1
: 1181 2097 1 IMPLICIT INPUTS:
: 1182 2098 1
: 1183 2099 1     The DEALLOC bit in the LUB
: 1184 2100 1
: 1185 2101 1 IMPLICIT OUTPUTS:
: 1186 2102 1
: 1187 2103 1     The ISB, and some other fields of the CCB
: 1188 2104 1     CCB             The restored CCB
: 1189 2105 1
: 1190 2106 1 SIDE EFFECTS:
: 1191 2107 1
: 1192 2108 1     Calls LIB$FREE_VM to free virtual memory.
: 1193 2109 1 --
: 1194 2110 1
: 1195 2111 2 BEGIN
: 1196 2112 2
: 1197 2113 2 EXTERNAL REGISTER
: 1198 2114 2     CCB : REF BLOCK [, BYTE];
: 1199 2115 2
: 1200 2116 2 LOCAL
: 1201 2117 2     PUSH : REF BLOCK [PUSH$K_LENGTH, BYTE] FIELD (PUSH_ITEM),
: 1202 2118 2     LUN;             ! Logical unit number being restored
: 1203 2119 2
: 1204 2120 2 +
: 1205 2121 2 Get an activation record off the I/O Active queue. It had better
: 1206 2122 2 be there.
: 1207 2123 2 -
: 1208 2124 2
: 1209 2125 2     IF (REMQUE (.OTSS$Q_IO_ACTIVE [0], PUSH)) THEN LIB$STOP (OTSS_FATINTERR);
: 1210 2126 2
: 1211 2127 2 +
: 1212 2128 2 Fetch the logical unit number associated with this record.
: 1213 2129 2 -
: 1214 2130 2     LUN = .PUSH [PUSH$W_LUN];
: 1215 2131 2 +
: 1216 2132 2 If this is a fake activation record, just store the LUN.
: 1217 2133 2 -
: 1218 2134 2

```

```

1219 2135 3      IF (.PUSH [PUSH$V_FAKE])
1220 2136 22      THEN
1221 2137 222      OTSS$L_CUR_LUN = .LUN
1222 2138 222      ELSE
1223 2139 222      BEGIN
1224 2140 222      +
1225 2141 222      | If this LUN does not have I/O in progress then something is very
1226 2142 222      | wrong.
1227 2143 222      -
1228 2144 222
1229 2145 222      IF ( NOT .OTSS$V_IOINPROG [.LUN - LUB$K_ILUN_MIN]) THEN LIB$STOP (OTSS$_FATINTERR);
1230 2146 222
1231 2147 222      +
1232 2148 222      | There was previous I/O. Restore the ISB, etc of the pushed unit.
1233 2149 222      | Because of ASTs, we must store OTSS$L_CUR_LUN before copying
1234 2150 222      | data from the I/O Active entry because only the LUN indicated by
1235 2151 222      | OTSS$L_CUR_LUN will get pushed.
1236 2152 222      -
1237 2153 222      OTSS$L_CUR_LUN = .LUN;
1238 2154 222      CCB = .OTSS$AA_LUB_TAB [.LUN, 0];
1239 2155 222      CCB = .CCB + (CCB [0, 0, 0, 0] - CCB [LUB$Q_QUEUE]);
1240 2156 222      OTSS$A_CUR_LUB = .CCB;
1241 2157 222      CCB [LUB$V_IO_ACTIVE] = .PUSH [PUSH$V_IO_ACT];
1242 2158 222      CCB [RAB$L_STS] = .PUSH [PUSH$L_STS];
1243 2159 222      CCB [RAB$L_STV] = .PUSH [PUSH$L_STV];
1244 2160 222      CCB [RAB$B_TMO] = .PUSH [PUSH$B_TMO];
1245 2161 222
1246 2162 222      IF (.PUSH [PUSH$V_PMT])
1247 2163 222      THEN
1248 2164 222      BEGIN
1249 2165 222      CCB [RAB$B_PSZ] = .PUSH [PUSH$B_PSZ];
1250 2166 222      CH$MOVE (.CCB [RAB$B_PSZ], PUSH [PUSH$T_PROMPT], .CCB [RAB$L_PBF]);
1251 2167 222      END;
1252 2168 222
1253 2169 222      CH$MOVE (ISB$K_ISB_LEN, PUSH [PUSH$X_ISB], .CCB - ISB$K_ISB_LEN - LUB$K_LUB_LEN);
1254 2170 222      +
1255 2171 222      | If the LUN has been marked for deallocation (which means that it
1256 2172 222      | has been closed but not deallocated yet because it has I/O in
1257 2173 222      | progress) then clear the statement type field so that all
1258 2174 222      | continued I/O will fail. The statement type must be set so that
1259 2175 222      | the owning language will get an error when I/O continues.
1260 2176 222      -
1261 2177 222
1262 2178 222      IF (.CCB [LUB$V_DEALLOC])
1263 2179 222      THEN
1264 2180 222
1265 2181 222      CASE .CCB [LUB$B_LANGUAGE] FROM LUB$K_LANG_MIN TO LUB$K_LANG_MAX OF
1266 2182 222      SET
1267 2183 222
1268 2184 222      [LUB$K_LANG_FOR] :
1269 2185 222      CCB [ISB$B_STTM_TYPE] = ISB$K_FORSTTYLO - 1;
1270 2186 222
1271 2187 222      [LUB$K_LANG_BAS] :
1272 2188 222      CCB [ISB$B_STTM_TYPE] = ISB$K_BASSTTYLO - 1;
1273 2189 222
1274 2190 222      [LUB$K_LANG_NONE] :
1275 2191 222      CCB [ISB$B_STTM_TYPE] = 0;

```


1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297

2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213

3
2
1

[OUTRANGE] :
LIB\$STOP (OTSS\$_FATINTERR);
TES;

END;

+ We are done with the item from the I/O Active List, free it.
-

BEGIN

LOCAL

FREE_VM_RESULT;

FREE_VM_RESULT = LIB\$FREE_VM (%REF (PUSH\$_LENGTH), PUSH);

IF (NOT .FREE_VM_RESULT) THEN LIB\$STOP (OTSS\$_FATINTERR);

END;

RETURN;

END;

! of routine POP_ACTIVE

03FC 00000 POP_ACTIVE:

		59	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	2078
		58	00000000G	00	9E	00009	MOVAB	OTSS\$_L_CUR_LUN, R9	
		57	00000000G	8F	D0	00010	MOVAB	LIB\$STOP, R8	
		5E		08	C2	00017	MOVL	#OTSS\$_FATINTERR, R7	
		AE	00000000'	FF	0F	0001A	SUBL2	#8, SP	
04				05	1C	00022	REMQUE	@OTSS\$_IO_ACTIVE, PUSH	2125
				57	DD	00024	BVC	1\$	
		68		01	FB	00026	PUSHL	R7	
		56	04	AE	D0	00029	CALLS	#1, LIB\$STOP	
		52	10	A6	32	0002D	MOVL	PUSH, R6	2130
05	14	A6		01	E1	00031	CVTWL	16(R6), LUN	
		69		52	D0	00036	BBC	#1, 20(R6), 2\$	2135
				7E	11	00039	MOVL	LUN, OTSS\$_L_CUR_LUN	2137
		53	08	A2	9E	0003B	BRB	8\$	
05	00000000G	00		53	E0	0003F	MOVAB	8(R2), R3	2145
				57	DD	00047	BBS	R3, OTSS\$_V_IOINPROG, 3\$	
		68		01	FB	00049	PUSHL	R7	
		69		52	D0	0004C	CALLS	#1, LIB\$STOP	
			00000000G0043	7F	0004F		MOVL	LUN, OTSS\$_L_CUR_LUN	2153
		5B		9E	D0	00056	PUSHAQ	OTSS\$_AA_LUB_TAB[R3]	2154
		5B		5B	C3	00059	MOVL	@(SP)+, CCB	
50		5B	58	A04B	9E	0005D	SUBL3	CCB, CCB, R0	2155
				5B	D0	00062	MOVAB	88(R0)[CCB], CCB	
	00000000G	00		5B	D0	00062	MOVL	CCB, OTSS\$_A_CUR_LUB	2156
FC	AB	01		01	14	A6	INSV	20(R6), #1, #1, -4(CCB)	2157
		08	AB	08	A6	7D	MOVQ	8(R6), 8(CCB)	2158
		1F	AB	13	A6	90	MOVB	19(R6), 31(CCB)	2160
		14	A6	02	E1	0007A	BBC	#2, 20(R6), 4\$	2162
		34	AB	12	A6	90	MOVB	18(R6), 52(CCB)	2165

OTSS\$CCB
1-057

K 8
16-Sep-1984 01:22:30
14-Sep-1984 12:39:38

VAX-11 Bliss-32 V4.0-742
[LIBRTL.SRC]OTSCCB.B32;1

Page 35
(10)

30	BB	15	50	34	AB	9A	00084	MOVZBL	52(CCB), R0	:	2166
FEE0	CB	65	A6	00BC	50	28	00088	MOVCL	R0, 21(R6), @48(CCB)	:	
	1D	FF	A6		8F	28	0008E	MOVCL	#188, 101(R6), -298(CCB)	:	2169
	02		AB	D8	04	E1	00097	BBC	#4, -1(CCB), 8\$:	2178
0014			00		AB	8F	0009C	CASEB	-40(CCB), #0, #2	:	2181
		000D		0014			000A1	.WORD	7\$-5\$,-	:	
									6\$-5\$,-	:	
									7\$-5\$:	
			68		57	DD	000A7	PUSHL	R7	:	2194
					01	FB	000A9	CALLS	#1, LIB\$STOP	:	
		FF71	CB		0B	11	000AC	BRB	8\$:	
					1A	90	000AE	MOVB	#26, -143(CCB)	:	2188
					04	11	000B3	BRB	8\$:	
				FF71	CB	94	000B5	CLRB	-143(CCB)	:	2191
				04	AE	9F	000B9	PUSHAB	PUSH	:	2207
		04	AE	0121	8F	3C	000BC	MOVZWL	#289, 4(SP)	:	
				04	AE	9F	000C2	PUSHAB	4(SP)	:	
		00000000G	00		02	FB	000C5	CALLS	#2, LIB\$FREE_VM	:	
			05		50	EB	000CC	BLBS	FREE_VM_RESULT, 9\$:	2209
			68		57	DD	000CF	PUSHL	R7	:	
					01	FB	000D1	CALLS	#1, LIB\$STOP	:	
					04	000D4	9\$:	RET		:	2213

; Routine Size: 213 bytes, Routine Base: _OTSS\$CODE + 03FF

; 1298 2214 1


```
1300 2215 1 GLOBAL ROUTINE OTSS$POP_CCB                ! Restore old CCB
1301 2216 1      : JSB_CB_POP NOVALUE =
1302 2217 1
1303 2218 1
1304 2219 1 ++
1305 2220 1 FUNCTIONAL DESCRIPTION:
1306 2221 1      Restore the I/O system to its state before the call to
1307 2222 1      PUSH_CCB. Clear LUB$V_IO_ACTIVE. If the I/O active list
1308 2223 1      is empty, clear OTSS$A_CUR_LUB, otherwise set it to
1309 2224 1      its previous value and restore its ISB, etc.
1310 2225 1
1311 2226 1      If virtual memory for a compiled format is allocated for this
1312 2227 1      ISB, it is freed.
1313 2228 1
1314 2229 1 CALLING SEQUENCE:
1315 2230 1      CALL OTSS$POP_CCB ( )
1316 2231 1
1317 2232 1 FORMAL PARAMETERS:
1318 2233 1
1319 2234 1      NONE
1320 2235 1
1321 2236 1 IMPLICIT INPUTS:
1322 2237 1
1323 2238 1      CCB
1324 2239 1      OTSS$AA_LUB_TAB
1325 2240 1      OTSS$Q_IO_ACTIVE
1326 2241 1
1327 2242 1 IMPLICIT OUTPUTS:
1328 2243 1
1329 2244 1      CCB
1330 2245 1      OTSS$Q_IO_ACTIVE      Set to previous LUB/ISB/RAB
1331 2246 1      LUB$V_IO_ACTIVE      Holds one fewer item
1332 2247 1                      Cleared to indicate I/O no longer active,
1333 2248 1                      but may be set by the pop from the
1334 2249 1                      I/O Active list.
1335 2250 1
1336 2251 1 SIDE EFFECTS:
1337 2252 1
1338 2253 1      May call LIB$FREE_VM to free virtual memory.
1339 2254 1 --
1340 2255 1
1341 2256 2 BEGIN
1342 2257 2
1343 2258 2 EXTERNAL REGISTER
1344 2259 2      CCB : REF BLOCK [, BYTE];
1345 2260 2
1346 2261 2 +
1347 2262 2 If the LUB has been marked for deallocation (by CLOSE) and there is
1348 2263 2 no I/O active, deallocate it. If there is I/O Active, the
1349 2264 2 deallocation must be deferred until after all of the I/O has completed
1350 2265 2 to insure that the continued I/O will get the "I/O continued to closed
1351 2266 2 file" error.
1352 2267 2 -
1353 2268 2
1354 2269 3 IF (.CCB [LUB$V_DEALLOC] AND ( NOT .CCB [ISB$V_RECURSIVE]))
1355 2270 2 THEN
1356 2271 2     DEALLOCATE ( )
```

```

1357 2272 2 ELSE
1358 2273 2 BEGIN
1359 2274 2 +
1360 2275 2 This is no longer the unit with I/O active.
1361 2276 2 -
1362 2277 2 CCB [LUB$V_IO_ACTIVE] = 0;
1363 2278 2 +
1364 2279 2 See if I/O will continue on this unit. It will continue if
1365 2280 2 ISB$V_RECURSIVE is set, which means that PUSH_CCB was called
1366 2281 2 with I/O in progress on this LUN. We make this test before
1367 2282 2 restoring the ISB because we may be restoring to the same
1368 2283 2 LUN, and the former I/O may be the top level of I/O for this
1369 2284 2 LUN, and if so it will have ISB$V_RECURSIVE clear.
1370 2285 2 -
1371 2286 2
1372 2287 2 IF ((.OTSS$L_LVL_CTR EQL 0) AND (.CCB [ISB$V_RECURSIVE])) THEN LIB$STOP (OTSS$_FATINTERR);
1373 2288 2
1374 2289 2 IF ( NOT .CCB [ISB$V_RECURSIVE])
1375 2290 2 THEN
1376 2291 2
1377 2292 2 IF (TESTBITCC (OTSS$V_IOINPROG [.CCB [LUB$W_LUN] - LUB$K_ILUN_MIN]))
1378 2293 2 THEN
1379 2294 2 LIB$STOP (OTSS$_FATINTERR);
1380 2295 2
1381 2296 2 END;
1382 2297 2
1383 2298 2 +
1384 2299 2 Since OTSS$V_IOINPROG may now be clear, our CCB may be deallocated, so
1385 2300 2 we cannot touch it again. For that matter, we may have deallocated
1386 2301 2 it ourselves above.
1387 2302 2
1388 2303 2 If there was previous I/O, restore it. Otherwise return to the idle
1389 2304 2 state.
1390 2305 2 -
1391 2306 2
1392 2307 2 IF (.OTSS$L_LVL_CTR NEQ 0)
1393 2308 2 THEN
1394 2309 2 POP_ACTIVE ()
1395 2310 2 ELSE
1396 2311 2 BEGIN
1397 2312 2 OTSS$A_CUR_LUB = 0;
1398 2313 2 OTSS$L_CUR_LUN = LUB$K_LUN_MAX + 1;
1399 2314 2 END;
1400 2315 2
1401 2316 2 +
1402 2317 2 Decrement the level counter. If we are at the top level the level
1403 2318 2 counter will go from 0 to -1.
1404 2319 2 -
1405 2320 2 OTSS$L_LVL_CTR = .OTSS$L_LVL_CTR - 1;
1406 2321 2 RETURN;
1407 2322 2 END;

```

! of routine OTSS\$POP_CCB

OB FF AB 04 E1 0000 OTSS\$POP_CCB::

	07	97	AB	E8	00005	BBC	#4, -1(CCB), 1\$: 2269
FE14	CF		00	FB	00009	BLBS	-105(CCB), 1\$: 2271
			3D	11	0000E	CALLS	#0, DEALLOCATE	: 2277
FC	AB		02	8A	00010	BRB	4\$: 2287
		00000000G	00	D5	00014	BICB2	#2, -4(CCB)	: 2289
			11	12	0001A	TSTL	OTSS\$SL_LVL_CTR	: 2292
	11	97	AB	E9	0001C	BNEQ	2\$: 2294
		00000000G	8F	DD	00020	BLBC	-105(CCB), 3\$: 2307
00000000G	00		01	FB	00026	PUSHL	#OTSS\$ FATINTERR	: 2309
	1C	97	AB	E8	0002D	CALLS	#1, LIB\$STOP	: 2312
	50	C6	AB	32	00031	BLBS	-105(CCB), 4\$: 2313
	50		08	C0	00035	CVTDL	-58(CCB), R0	: 2320
OD 00000000G	00		50	E4	00038	ADDL2	#8, R0	: 2322
		00000000G	8F	DD	00040	BBSC	R0, OTSS\$V IOINPROG, 4\$	
00000000G	00		01	FB	00046	PUSHL	#OTSS\$ FATINTERR	
		00000000G	00	D5	0004D	CALLS	#1, LIB\$STOP	
			07	13	00053	TSTL	OTSS\$SL_LVL_CTR	
FED1	CF		00	FB	00055	BEQL	5\$	
		00000000G	0E	11	0005A	CALLS	#0, POP_ACTIVE	
			00	D4	0005C	BRB	6\$	
00000000G	00	78	8F	9A	00062	CLRL	OTSS\$A CUR_LUB	
		00000000G	00	D7	0006A	MOVZBL	#120, OTSS\$SL_CUR_LUN	
			05	00070	DECL	OTSS\$SL_LVL_CTR		
					RSB			

; Routine Size: 113 bytes, Routine Base: _OTSS\$CODE + 04D4

: 1408 2323 1
: 1409 2324 1 END
: 1410 2325 1
: 1411 2326 0 ELUDOM

!End of module OTSS\$CCB

PSECT SUMMARY

Name	Bytes	Attributes
_OTSS\$DATA	8	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
_OTSS\$CODE	1349	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	20	0	581	00:00.8

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LISS:OTSCCB/OBJ=OBJ\$:OTSCCB MSRC\$:OTSCCB/UPDATE=(ENH\$:OTSCCB)

: Size: 1349 code + 8 data bytes
: Run Time: 00:21.3
: Elapsed Time: 01:31.4
: Lines/CPU Min: 6567
: Lexemes/CPU-Min: 37228
: Memory Used: 185 pages
: Compilation Complete

0211 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

